Ansible 2.15 Full Offline Documentation

Release stable 2.15

Witty Enthusiasm 318

May 11, 2023

ANSIBLE GETTING STARTED

CHAPTER

ONE

ABOUT ANSIBLE

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Ansible's main goals are simplicity and ease-of-use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with other transports and pull modes as alternatives), and a language that is designed around auditability by humans–even those not familiar with the program.

We believe simplicity is relevant to all sizes of environments, so we design for busy users of all types: developers, sysadmins, release engineers, IT managers, and everyone in between. Ansible is appropriate for managing all environments, from small setups with a handful of instances to enterprise environments with many thousands of instances.

You can learn more at AnsibleFest, the annual event for all Ansible contributors, users, and customers hosted by Red Hat. AnsibleFest is the place to connect with others, learn new skills, and find a new friend to automate with.

Ansible manages machines in an agent-less manner. There is never a question of how to upgrade remote daemons or the problem of not being able to manage systems because daemons are uninstalled. Also, security exposure is greatly reduced because Ansible uses OpenSSH — the open source connectivity tool for remote login with the SSH (Secure Shell) protocol.

Ansible is decentralized–it relies on your existing OS credentials to control access to remote machines. And if needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

This documentation covers the version of Ansible noted in the upper left corner of this page. We maintain multiple versions of Ansible and the Ansible documentation, so please be sure you are using the documentation version that covers the version of Ansible you are using. For recent features, we note the version of Ansible where the feature was added.

Ansible releases a new major release approximately twice a year. The core application evolves somewhat conservatively, valuing simplicity in language design and setup. Contributors develop and change modules and plugins hosted in collections since version 2.10 much more quickly.

1.1 Getting started with Ansible

Ansible automates the management of remote systems and controls their desired state. A basic Ansible environment has three main components:

Control node
: A system on which Ansible is installed. You run Ansible commands such as `ansible` or `ansible-inventory` on a control node.

Managed node
: A remote system, or host, that Ansible controls.

Inventory
A list of managed nodes that are logically organized. You create an inventory on the control node to describe host deployments to Ansible.

Ready to start using Ansible? Complete the following steps to get up and running:

1. Install Ansible. Visit the *installation guide* for complete details.

```
python3 -m pip install --user ansible
```

2. Create an inventory by adding the IP address or fully qualified domain name (FQDN) of one or more remote systems to `/etc/ansible/hosts`. The following example adds the IP addresses of three virtual machines in KVM:

```
[myvirtualmachines]
192.0.2.50
192.0.2.51
192.0.2.52
```

3. Verify the hosts in your inventory.

```
ansible all --list-hosts
```

```
hosts (1):
  192.0.2.50
  192.0.2.51
  192.0.2.52
```

4. Set up SSH connections so Ansible can connect to the managed nodes.

   a. Add your public SSH key to the `authorized_keys` file on each remote system.

   b. Test the SSH connections, for example:

```
ssh username@192.0.2.50
```

If the username on the control node is different on the host, you need to pass the `-u` option with the `ansible` command.

5. Ping the managed nodes.

```
ansible all -m ping
```

```
192.0.2.50 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
    }
192.0.2.51 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
```

(continues on next page)

(continued from previous page)

```
    "ping": "pong"
    }
192.0.2.52 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
    }
```

Congratulations! You are now using Ansible. Continue by *learning how to build an inventory*.

See also:

Ansible Demos
    Demonstrations of different Ansible usecases

Ansible Labs
    Labs to provide further knowledge on different topics

Mailing List
    Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
    How to join Ansible chat channels

1.1.1 Building an inventory

Inventories organize managed nodes in centralized files that provide Ansible with system information and network locations. Using an inventory file, Ansible can manage a large number of hosts with a single command. Inventories also help you use Ansible more efficiently by reducing the number of command-line options you need to specify. For example, inventories usually contain the SSH user so you do not need to include the `-u` flag when running Ansible commands.

In the previous section, you added managed nodes directly to the `/etc/ansible/hosts` file. Now let's create an inventory file that you can add to source control for flexibility, reuse, and for sharing with other users.

Note: Inventory files can be in `INI` or `YAML` format. For demonstration purposes, this section uses `YAML` format only.

Complete the following steps:

1. Open a terminal window on your control node.
2. Create a new inventory file named `inventory.yaml` in any directory and open it for editing.
3. Add a new group for your hosts then specify the IP address or fully qualified domain name (FQDN) of each managed node with the `ansible_host` field. The following example adds the IP addresses of three virtual machines in KVM:

```yaml
virtualmachines:
  hosts:
    vm01:
      ansible_host: 192.0.2.50
    vm02:
      ansible_host: 192.0.2.51
```

(continues on next page)

(continued from previous page)

```
    vm03:
      ansible_host: 192.0.2.52
```

4. Verify your inventory. If you created your inventory in a directory other than your home directory, specify the full path with the `-i` option.

```
ansible-inventory -i inventory.yaml --list
```

5. Ping the managed nodes in your inventory. In this example, the group name is `virtualmachines` which you can specify with the `ansible` command instead of `all`.

```
ansible virtualmachines -m ping -i inventory.yaml
```

```
vm03 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
vm02 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
vm01 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Congratulations! You have successfully built an inventory.

Tips for building inventories

- Ensure that group names are meaningful and unique. Group names are also case sensitive.
- Avoid spaces, hyphens, and preceding numbers (use `floor_19`, not `19th_floor`) in group names.
- Group hosts in your inventory logically according to their **What**, **Where**, and **When**.

  What

  Group hosts according to the topology, for example: db, web, leaf, spine.

  Where

  Group hosts by geographic location, for example: datacenter, region, floor, building.

  When

  Group hosts by stage, for example: development, test, staging, production.

Use metagroups

Create a metagroup that organizes multiple groups in your inventory with the following syntax:

```
metagroupname:
  children:
```

The following inventory illustrates a basic structure for a data center. This example inventory contains a `network` metagroup that includes all network devices and a `datacenter` metagroup that includes the `network` group and all webservers.

```
leafs:
  hosts:
    leaf01:
      ansible_host: 192.0.2.100
    leaf02:
      ansible_host: 192.0.2.110

spines:
  hosts:
    spine01:
      ansible_host: 192.0.2.120
    spine02:
      ansible_host: 192.0.2.130

network:
  children:
    leafs:
    spines:

webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
    webserver02:
      ansible_host: 192.0.2.150

datacenter:
  children:
    network:
    webservers:
```

Create variables

Variables set values for managed nodes, such as the IP address, FQDN, operating system, and SSH user, so you do not need to pass them when running Ansible commands.

Variables can apply to specific hosts.

```
webservers:
  hosts:
    webserver01:
```

(continues on next page)

(continued from previous page)

```yaml
      ansible_host: 192.0.2.140
      http_port: 80
    webserver02:
      ansible_host: 192.0.2.150
      http_port: 443
```

Variables can also apply to all hosts in a group.

```yaml
webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
      http_port: 80
    webserver02:
      ansible_host: 192.0.2.150
      http_port: 443
  vars:
    ansible_user: my_server_user
```

Now that you know how to build an inventory, continue by *learning how to create a playbook*.

See also:

How to build your inventory
: Learn more about inventories in `YAML` or `INI` format.

Adding variables to inventory
: Find out more about inventory variables and their syntax.

Ansible Vault
: Find out how to encrypt sensitive content in your inventory such as passwords and keys.

1.1.2 Creating a playbook

Playbooks are automation blueprints, in `YAML` format, that Ansible uses to deploy and configure managed nodes.

Playbook
: A list of plays that define the order in which Ansible performs operations, from top to bottom, to achieve an overall goal.

Play
: An ordered list of tasks that maps to managed nodes in an inventory.

Task
: A list of one or more modules that defines the operations that Ansible performs.

Module
: A unit of code or binary that Ansible runs on managed nodes. Ansible modules are grouped in collections with a *Fully Qualified Collection Name (FQCN)* for each module.

In the previous section, you used the `ansible` command to ping hosts in your inventory. Now let's create a playbook that pings your hosts and also prints a "Hello world" message.

Complete the following steps:

1. Open a terminal window on your control node.
2. Create a new playbook file named `playbook.yaml` in any directory and open it for editing.

3. Add the following content to `playbook.yaml`:

```
- name: My first play
  hosts: virtualmachines
  tasks:
   - name: Ping my hosts
     ansible.builtin.ping:
   - name: Print message
     ansible.builtin.debug:
       msg: Hello world
```

4. Run your playbook.

```
ansible-playbook -i inventory.yaml playbook.yaml
```

Ansible returns the following output:

```
PLAY [My first play]␣
↪**********************************************************************

TASK [Gathering Facts]␣
↪********************************************************************
ok: [vm01]
ok: [vm02]
ok: [vm03]

TASK [Ping my hosts]␣
↪**********************************************************************
ok: [vm01]
ok: [vm02]
ok: [vm03]

TASK [Print message]␣
↪**********************************************************************
ok: [vm01] => {
    "msg": "Hello world"
}
ok: [vm02] => {
    "msg": "Hello world"
}
ok: [vm03] => {
    "msg": "Hello world"
}

PLAY RECAP␣
↪********************************************************************************
vm01: ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ␣
↪ignored=0
vm02: ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ␣
↪ignored=0
vm03: ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ␣
↪ignored=0
```

In this output you can see:

- The names that you give the play and each task. You should always use descriptive names that make it easy to verify and troubleshoot playbooks.
- The `Gather Facts` task runs implicitly. By default Ansible gathers information about your inventory that it can use in the playbook.
- The status of each task. Each task has a status of `ok` which means it ran successfully.
- The play recap that summarizes results of all tasks in the playbook per host. In this example, there are three tasks so `ok=3` indicates that each task ran successfully.

Congratulations! You have just created your first Ansible playbook.

See also:

Ansible playbooks
Start building playbooks for real world scenarios.

Working with playbooks
Go into more detail with Ansible playbooks.

Ansible tips and tricks
Get tips and tricks for using playbooks.

Discovering variables: facts and magic variables
Learn more about the `gather_facts` keyword in playbooks.

1.1.3 Ansible concepts

These concepts are common to all uses of Ansible. You should understand them before using Ansible or reading the documentation.

Control node

The machine from which you run the Ansible CLI tools (`ansible-playbook` , `ansible`, `ansible-vault` and others). You can use any computer that meets the software requirements as a control node - laptops, shared desktops, and servers can all run Ansible. Multiple control nodes are possible, but Ansible itself does not coordinate across them, see `AAP` for such features.

Managed nodes

Also referred to as 'hosts', these are the target devices (servers, network appliances or any computer) you aim to manage with Ansible. Ansible is not normally installed on managed nodes, unless you are using `ansible-pull`, but this is rare and not the recommended setup.

Inventory

A list of managed nodes provided by one or more 'inventory sources'. Your inventory can specify information specific to each node, like IP address. It is also used for assigning groups, that both allow for node selection in the Play and bulk variable assignment. To learn more about inventory, see *the Working with Inventory* section. Sometimes an inventory source file is also referred to as a 'hostfile'.

Playbooks

They contain Plays (which are the basic unit of Ansible execution). This is both an 'execution concept' and how we describe the files on which `ansible-playbook` operates. Playbooks are written in YAML and are easy to read, write, share and understand. To learn more about playbooks, see *Ansible playbooks*.

Plays

The main context for Ansible execution, this playbook object maps managed nodes (hosts) to tasks. The Play contains variables, roles and an ordered lists of tasks and can be run repeatedly. It basically consists of an implicit loop over the mapped hosts and tasks and defines how to iterate over them.

Roles

A limited distribution of reusable Ansible content (tasks, handlers, variables, plugins, templates and files) for use inside of a Play. To use any Role resource, the Role itself must be imported into the Play.

Tasks

The definition of an 'action' to be applied to the managed host. Tasks must always be contained in a Play, directly or indirectly (Role, or imported/included task list file). You can execute a single task once with an ad hoc command using `ansible` or `ansible-console` (both create a virtual Play).

Handlers

A special form of a Task, that only executes when notified by a previous task which resulted in a 'changed' status.

Modules

The code or binaries that Ansible copies to and executes on each managed node (when needed) to accomplish the action defined in each Task. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. You can invoke a single module with a task, or invoke several different modules in a playbook. Ansible modules are grouped in collections. For an idea of how many collections Ansible includes, see the Collection Index.

Plugins

Pieces of code that expand Ansible's core capabilities, they can control how you connect to a managed node (connection plugins), manipulate data (filter plugins) and even control what is displayed in the console (callback plugins). See *Working with plugins* for details.

Collections

A format in which Ansible content is distributed that can contain playbooks, roles, modules, and plugins. You can install and use collections through Ansible Galaxy. To learn more about collections, see *Using Ansible collections*. Collection resources can be used independently and discretely from each other.

AAP

Short for 'Ansible Automation Platform'. This is a product that includes enterprise level features and integrates many tools of the Ansible ecosystem: ansible-core, awx, galaxyNG, and so on.

1.2 Installation Guide

Welcome to the Ansible Installation Guide!

1.2.1 Installing Ansible

Ansible is an agentless automation tool that you install on a single host (referred to as the control node). From the control node, Ansible can manage an entire fleet of machines and other devices (referred to as managed nodes) remotely with SSH, Powershell remoting, and numerous other transports, all from a simple command-line interface with no databases or daemons required.

Control node requirements

For your *control* node (the machine that runs Ansible), you can use nearly any UNIX-like machine with Python 3.9 or newer installed. This includes Red Hat, Debian, Ubuntu, macOS, BSDs, and Windows under a Windows Subsystem for Linux (WSL) distribution. Windows without WSL is not natively supported as a control node; see Matt Davis' blog post for more information.

Managed node requirements

The *managed* node (the machine that Ansible is managing) does not require Ansible to be installed, but requires Python 2.7, or Python 3.5 - 3.11 to run Ansible library code. The managed node also needs a user account that can SSH to the node with an interactive POSIX shell.

Note: Network modules are an exception and do not require Python on the managed device. See Network modules.

Node requirement summary

The table below lists the current and historical versions of Python required on control and managed nodes.

ansible-core Version	Control node Python	Managed node Python
2.11	Python 2.7, Python 3.5 - 3.9 *[†]*	Python 2.6 - 2.7, Python 3.5 - 3.9
2.12	Python 3.8 - 3.10	Python 2.6 - 2.7, Python 3.5 - 3.10
2.13	Python 3.8 - 3.10	Python 2.7, Python 3.5 - 3.10
2.14	Python 3.9 - 3.11	Python 2.7, Python 3.5 - 3.11

[†]: Has a soft requirement of Python 3.8 as not packaged for older versions

Selecting an Ansible package and version to install

Ansible's community packages are distributed in two ways: a minimalist language and runtime package called `ansible-core`, and a much larger "batteries included" package called `ansible`, which adds a community-curated selection of *Ansible Collections* for automating a wide variety of devices. Choose the package that fits your needs; The following instructions use `ansible`, but you can substitute `ansible-core` if you prefer to start with a more minimal package and separately install only the Ansible Collections you require. The `ansible` or `ansible-core` packages may be available in your operating systems package manager, and you are free to install these packages with your preferred method. These installation instructions only cover the officially supported means of installing the python package with `pip`.

See the *Ansible package release status table* for the `ansible-core` version included in the package.

Installing and upgrading Ansible

Locating Python

Locate and remember the path to the Python interpreter you wish to use to run Ansible. The following instructions refer to this Python as `python3`. For example, if you've determined that you want the Python at `/usr/bin/python3.9` to be the one that you'll install Ansible under, specify that instead of `python3`.

Ensuring `pip` is available

To verify whether `pip` is already installed for your preferred Python:

```
$ python3 -m pip -V
```

If all is well, you should see something like the following:

```
$ python3 -m pip -V
pip 21.0.1 from /usr/lib/python3.9/site-packages/pip (python 3.9)
```

If so, `pip` is available, and you can move on to the *next step*.

If you see an error like `No module named pip`, you'll need to install `pip` under your chosen Python interpreter before proceeding. This may mean installing an additional OS package (for example, `python3-pip`), or installing the latest `pip` directly from the Python Packaging Authority by running the following:

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
$ python3 get-pip.py --user
```

You may need to perform some additional configuration before you are able to run Ansible. See the Python documentation on installing to the user site for more information.

Installing Ansible

Use pip in your selected Python environment to install the Ansible package of your choice for the current user:

```
$ python3 -m pip install --user ansible
```

Alternately, you can install a specific version of ansible-core in this Python environment:

```
$ python3 -m pip install --user ansible-core==2.12.3
```

Upgrading Ansible

To upgrade an existing Ansible installation in this Python environment to the latest released version, simply add --upgrade to the command above:

```
$ python3 -m pip install --upgrade --user ansible
```

Confirming your installation

You can test that Ansible is installed correctly by checking the version:

```
$ ansible --version
```

The version displayed by this command is for the associated ansible-core package that has been installed.

To check the version of the ansible package that has been installed:

```
$ python3 -m pip show ansible
```

Installing for development

If you are testing new features, fixing bugs, or otherwise working with the development team on changes to the core code, you can install and run the source from GitHub.

Note: You should only install and run the devel branch if you are modifying ansible-core or trying out features under development. This is a rapidly changing source of code and can become unstable at any point.

For more information on getting involved in the Ansible project, see the *Ansible Community Guide*. For more information on creating Ansible modules and Collections, see the *Developer Guide*.

Installing `devel` from GitHub with `pip`

You can install the `devel` branch of `ansible-core` directly from GitHub with `pip`:

```
$ python3 -m pip install --user https://github.com/ansible/ansible/archive/devel.tar.gz
```

You can replace `devel` in the URL mentioned above, with any other branch or tag on GitHub to install older versions of Ansible, tagged alpha or beta versions, and release candidates.

Running the `devel` branch from a clone

`ansible-core` is easy to run from source. You do not need `root` permissions to use it and there is no software to actually install. No daemons or database setup are required.

1. Clone the `ansible-core` repository

   ```
   $ git clone https://github.com/ansible/ansible.git
   $ cd ./ansible
   ```

2. Setup the Ansible environment

   - Using Bash

     ```
     $ source ./hacking/env-setup
     ```

   - Using Fish

     ```
     $ source ./hacking/env-setup.fish
     ```

   - To suppress spurious warnings/errors, use `-q`

     ```
     $ source ./hacking/env-setup -q
     ```

3. Install Python dependencies

   ```
   $ python3 -m pip install --user -r ./requirements.txt
   ```

4. Update the `devel` branch of `ansible-core` on your local machine

   Use pull-with-rebase so any local changes are replayed.

   ```
   $ git pull --rebase
   ```

Adding Ansible command shell completion

You can add shell completion of the Ansible command line utilities by installing an optional dependency called `argcomplete`. `argcomplete` supports bash, and has limited support for zsh and tcsh.

For more information about installation and configuration, see the argcomplete documentation.

Installing `argcomplete`

```
$ python3 -m pip install --user argcomplete
```

Configuring `argcomplete`

There are 2 ways to configure `argcomplete` to allow shell completion of the Ansible command line utilities: globally or per command.

Global configuration

Global completion requires bash 4.2.

```
$ activate-global-python-argcomplete --user
```

This will write a bash completion file to a user location. Use `--dest` to change the location or `sudo` to set up the completion globally.

Per command configuration

If you do not have bash 4.2, you must register each script independently.

```
$ eval $(register-python-argcomplete ansible)
$ eval $(register-python-argcomplete ansible-config)
$ eval $(register-python-argcomplete ansible-console)
$ eval $(register-python-argcomplete ansible-doc)
$ eval $(register-python-argcomplete ansible-galaxy)
$ eval $(register-python-argcomplete ansible-inventory)
$ eval $(register-python-argcomplete ansible-playbook)
$ eval $(register-python-argcomplete ansible-pull)
$ eval $(register-python-argcomplete ansible-vault)
```

You should place the above commands into your shells profile file such as `~/.profile` or `~/.bash_profile`.

Using `argcomplete` with zsh or tcsh

See the argcomplete documentation.

See also:

Introduction to ad hoc commands
    Examples of basic commands

Working with playbooks
    Learning ansible's configuration management language

How do I handle the package dependencies required by Ansible package dependencies during Ansible installation ?
    Ansible Installation related to FAQs

Mailing List
    Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.2.2 Installing Ansible on specific operating systems

Note: These instructions are provided by their respective communities. Any bugs/issues should be filed with that community to update these instructions. Ansible maintains only the `pip install` instructions.

The `ansible` package can always be *installed from PyPI using pip* on most systems but it is also packaged and maintained by the community for a variety of Linux distributions.

The following instructions will guide you through installing the `ansible` package with your preferred distribution's package manager.

Note: For maintainers who wish to add distributions to this guide, installation instructions are included here only for distributions with a reasonably up-to-date version of `ansible`. The distribution MUST ensure that `ansible-core` and `ansible` versions are kept in sync to the extent that the distribution build system allows. Maintainers MUST include a way to contact them with their instructions here and are encouraged to join the Ansible Packaging Matrix room.

Installing Ansible on Fedora Linux

To install the batteries included `ansible` package on Fedora run

```
$ sudo dnf install ansible
```

If you prefer to install the minimal `ansible-core` package run

```
$ sudo dnf install ansible-core
```

Several Ansible collections are also available from the Fedora repositories as standalone packages that users can install alongside `ansible-core`. For example, to install the `community.general` collection run

```
$ sudo dnf install ansible-collection-community-general
```

See the Fedora Packages index for a full list of Ansible collections packaged in Fedora.

Please file a bug against the `Fedora` product in Red Hat Bugzilla to reach the package maintainers.

Installing Ansible from EPEL

Users of CentOS Stream, Almalinux, Rocky Linux, and related distributions can install `ansible` or Ansible collections from the community maintained EPEL (Extra Packages for Enterprise Linux) repository.

After enabling the EPEL repository, users can use the same `dnf` commands as for Fedora Linux.

Please file a bug against the `Fedora EPEL` product in Red Hat Bugzilla to reach the package maintainers.

Installing Ansible on OpenSUSE Tumbleweed/Leap

```
$ sudo zypper install ansible
```

See *OpenSUSE Support Portal <https://en.opensuse.org/Portal:Support>* for additional help with Ansible on OpenSUSE.

Installing Ansible on Ubuntu

Ubuntu builds are available in a PPA here.

To configure the PPA on your system and install Ansible run these commands:

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible
```

Note: On older Ubuntu distributions, "software-properties-common" is called "python-software-properties". You may want to use `apt-get` rather than `apt` in older versions. Also, be aware that only newer distributions (that is, 18.04, 18.10, and later) have a `-u` or `--update` flag. Adjust your script as needed.

Installing Ansible on Debian

Debian users can use the same source as the Ubuntu PPA (using the following table).

Debian		Ubuntu
Debian 11 (Bullseye)	->	Ubuntu 20.04 (Focal)
Debian 10 (Buster)	->	Ubuntu 18.04 (Bionic)

Note: Ansible releases are only built for Ubuntu 18.04 (Bionic) or later releases.

Add the following line to `/etc/apt/sources.list` or `/etc/apt/sources.list.d/ansible.list`:

```
deb http://ppa.launchpad.net/ansible/ansible/ubuntu MATCHING_UBUNTU_CODENAME_HERE main
```

Example for Debian 11 (Bullseye)

```
deb http://ppa.launchpad.net/ansible/ansible/ubuntu focal main
```

Then run these commands:

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367
$ sudo apt update
$ sudo apt install ansible
```

Installing Ansible on Windows

You cannot use a Windows system for the Ansible control node. See *Can Ansible run on Windows?*

See also:

Installing Ansible on Arch Linux
Distro-specific installation on Arch Linux

Installing Ansible on Clear Linux
Distro-specific installation on Clear Linux

1.2.3 Configuring Ansible

Topics

This topic describes how to control Ansible settings.

Configuration file

Certain settings in Ansible are adjustable via a configuration file (ansible.cfg). The stock configuration should be sufficient for most users, but there may be reasons you would want to change them. Paths where configuration file is searched are listed in *reference documentation*.

Getting the latest configuration

If installing Ansible from a package manager, the latest `ansible.cfg` file should be present in `/etc/ansible`, possibly as a `.rpmnew` file (or other) as appropriate in the case of updates.

If you installed Ansible from pip or from source, you may want to create this file in order to override default settings in Ansible.

An example file is available on GitHub.

For more details and a full listing of available configurations go to *configuration_settings*. Starting with Ansible version 2.4, you can use the *ansible-config* command line utility to list your available options and inspect the current values.

For in-depth details, see *Ansible Configuration Settings*.

Environmental configuration

Ansible also allows configuration of settings using environment variables. If these environment variables are set, they will override any setting loaded from the configuration file.

You can get a full listing of available environment variables for configuring core functionality from *Ansible Configuration Settings*, and for configuring plugins in collections from list_of_collection_env_vars.

Command line options

Not all configuration options are present in the command line, just the ones deemed most useful or common. Settings in the command line will override those passed through the configuration file and the environment.

The full list of options available is in *ansible-playbook* and *ansible*.

1.3 Ansible Porting Guides

This section lists porting guides that can help you in updating playbooks, plugins and other parts of your Ansible infrastructure from one version of Ansible to the next.

1.3.1 Ansible 8 Porting Guide

Ansible 8 is based on Ansible-core 2.15.

We suggest you read this page along with the Ansible 8 Changelog to understand what updates you may need to make.

Playbook

No notable changes

Command Line

- The return code of `ansible-galaxy search` is now 0 instead of 1 and the stdout is empty when results are empty to align with other `ansible-galaxy` commands.

Deprecated

- Providing a list of dictionaries to `vars:` is deprecated in favor of supplying a dictionary.

  Instead of:

  ```
  vars:
    - var1: foo
    - var2: bar
  ```

  Use:

  ```
  vars:
    var1: foo
    var2: bar
  ```

Modules

No notable changes

Modules removed

The following modules no longer exist:

- No notable changes

Deprecation notices

No notable changes

Noteworthy module changes

No notable changes

Plugins

No notable changes

Porting custom scripts

No notable changes

Networking

No notable changes

Porting Guide for v8.0.0a1

Added Collections

- dellemc.powerflex (version 1.6.0)
- dellemc.unity (version 1.6.0)
- grafana.grafana (version 2.0.0)
- microsoft.ad (version 1.0.0)
- servicenow.servicenow (version 1.0.6)

Known Issues

Ansible-core

- ansible-test - Additional configuration may be required for certain container host and container combinations. Further details are available in the testing documentation.
- ansible-test - Custom containers with `VOLUME` instructions may be unable to start, when previously the containers started correctly. Remove the `VOLUME` instructions to resolve the issue. Containers with this condition will cause `ansible-test` to emit a warning.
- ansible-test - Systems with Podman networking issues may be unable to run containers, when previously the issue went unreported. Correct the networking issues to continue using `ansible-test` with Podman.
- ansible-test - Unit tests for collections do not support `pytest` assertion rewriting on Python 2.7.
- ansible-test - Using Docker on systems with SELinux may require setting SELinux to permissive mode. Podman should work with SELinux in enforcing mode.
- dnf5 - The DNF5 package manager currently does not provide all functionality to ensure feature parity between the existing `dnf` and the new `dnf5` module. As a result the following `dnf5` options are effectively a no-op: `cacheonly`, `enable_plugin`, `disable_plugin` and `lock_timeout`.

cisco.meraki

- meraki_network - Updated documentation for *local_status_page_enabled* and *remote_status_page_enabled* as these no longer work.

community.routeros

- api_modify - when limits for entries in `queue tree` are defined as human readable - for example `25M` -, the configuration will be correctly set in ROS, but the module will indicate the item is changed on every run even when there was no change done. This is caused by the ROS API which returns the number in bytes - for example `25000000` (which is inconsistent with the CLI behavior). In order to mitigate that, the limits have to be defined in bytes (those will still appear as human readable in the ROS CLI) (https://github.com/ansible-collections/community.routeros/pull/131).
- api_modify, api_info - `routing ospf area`, `routing ospf area range`, `routing ospf instance`, `routing ospf interface-template` paths are not fully implemeted for ROS6 due to the significat changes between ROS6 and ROS7 (https://github.com/ansible-collections/community.routeros/pull/131).

dellemc.openmanage

- idrac_firmware - Issue(249879) - Firmware update of iDRAC9-based Servers fails if SOCKS proxy with authentication is used.
- idrac_os_deployment- Issue(260496) - OS installation will support only NFS and CIFS share to store the custom ISO in the destination_path, HTTP/HTTPS/FTP not supported
- idrac_redfish_storage_contoller - Issue(256164) - If incorrect value is provided for one of the attributes in the provided attribute list for controller configuration, then this module does not exit with error.
- idrac_user - Issue(192043) The module may error out with the message `Unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- idrac_user - Issue(192043) The module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_application_alerts_syslog - Issue(215374) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the following parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_power_settings - Issue(212679) - The module displays the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_inventory - Issue(256257) - All hosts are not retrieved for `Modular System` group and corresponding child groups.
- ome_inventory - Issue(256589) - All hosts are not retrieved for `Custom Groups` group and corresponding child groups.

- ome_inventory - Issue(256593) - All hosts are not retrieved for `PLUGIN GROUPS` group and corresponding child groups.
- ome_smart_fabric_uplink - Issue(186024) - Despite the module supported by OpenManage Enterprise Modular, it does not allow the creation of multiple uplinks of the same name. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.
- ome_smart_fabric_uplink - Issue(186024) - The module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Breaking Changes

Ansible-core

- ansible-doc - no longer treat plugins in collections whose name starts with `_` as deprecated (https://github.com/ansible/ansible/pull/79217).
- ansible-test - Integration tests which depend on specific file permissions when running in an ansible-test managed host environment may require changes. Tests that require permissions other than `755` or `644` may need to be updated to set the necessary permissions as part of the test run.
- ansible-test - The `vcenter` test plugin now defaults to using a user-provided static configuration instead of the `govcsim` simulator for collections. Set the `ANSIBLE_VCSIM_CONTAINER` environment variable to `govcsim` to use the simulator. Keep in mind that the simulator is deprecated and will be removed in a future release.
- ansible-test sanity - previously plugins and modules in collections whose name started with `_` were treated as deprecated, even when they were not marked as deprecated in `meta/runtime.yml`. This is no longer the case (https://github.com/ansible/ansible/pull/79362).
- ansible-test validate-modules - Removed the `missing-python-doc` error code in validate modules, `missing-documentation` is used instead for missing PowerShell module documentation.

ansible.netcommon

- NetworkConnectionBase now inherits from PersistentConnectionBase in ansible.utils. As a result, the minimum ansible.utils version has increased to 2.7.0.
- NetworkTemplate is no longer importable from ansible_collections.ansible.netcommon.plugins.module_utils.network.common and should now be found at its proper location ansible_collections.ansible.netcommon.plugins.module_utils.network.common.rm_b
- ResourceModule is no longer importable from ansible_collections.ansible.netcommon.plugins.module_utils.network.common and should now be found at its proper location ansible_collections.ansible.netcommon.plugins.module_utils.network.common.rm_b
- VALID_MASKS, is_masklen, is_netmask, to_bits, to_ipv6_network, to_masklen, to_netmask, and to_subnet are no longer importable from ansible_collections.ansible.netcommon.plugins.module_utils.network.common.utils and should now be found at their proper location ansible.module_utils.common.network

community.general

- ModuleHelper module utils - when the module sets output variables named `msg`, `exception`, `output`, `vars`, or `changed`, the actual output will prefix those names with `_` (underscore symbol) only when they clash with output variables generated by ModuleHelper itself, which only occurs when handling exceptions. Please note that this breaking change does not require a new major release since before this release, it was not possible to add such variables to the output due to a bug (https://github.com/ansible-collections/community.general/pull/5765).

hetzner.hcloud

- inventory plugin - Python v3.5+ is now required.

Major Changes

Ansible-core

- ansible-test - Docker Desktop on WSL2 is now supported (additional configuration required).
- ansible-test - Docker and Podman are now supported on hosts with cgroup v2 unified. Previously only cgroup v1 and cgroup v2 hybrid were supported.
- ansible-test - Podman now works on container hosts without systemd. Previously only some containers worked, while others required rootfull or rootless Podman, but would not work with both. Some containers did not work at all.
- ansible-test - Podman on WSL2 is now supported.
- ansible-test - When additional cgroup setup is required on the container host, this will be automatically detected. Instructions on how to configure the host will be provided in the error message shown.

ansible.windows

- Set the minimum Ansible version supported by this collection to Ansible 2.12

chocolatey.chocolatey

- win_chocolatey - Allow users to select the TLS versions used for bootstrapping Chocolatey installation.

cisco.iosxr

- iosxr_l3_interfaces - fix issue in ipv4 address formatting. (https://github.com/ansible-collections/cisco.iosxr/issues/311).

cisco.meraki

- meraki_mr_l7_firewall - New module
- meraki_webhook_payload_template - New module

community.hrobot

- firewall - Hetzner added output rules support to the firewall. This change unfortunately means that using old versions of the firewall module will always set the output rule list to empty, thus disallowing the server to send out packets (https://github.com/ansible-collections/community.hrobot/issues/75, https://github.com/ansible-collections/community.hrobot/pull/76).

community.vmware

- Use true/false (lowercase) for boolean values in documentation and examples (https://github.com/ansible-collections/community.vmware/issues/1660).

community.zabbix

- all modules are opting away from zabbix-api and using httpapi ansible.netcommon plugin. We will support zabbix-api for backwards compatibility until next major release. See our README.md for more information about how to migrate
- zabbix_agent and zabbix_proxy roles are opting away from zabbix-api and use httpapi ansible.netcommon plugin. We will support zabbix-api for backwards compatibility until next major release. See our README.md for more information about how to migrate

containers.podman

- New become plugin - podman_unshare
- Podman generate systemd module

dellemc.openmanage

- Rebranded from Dell EMC to Dell.
- Support for IPv6 address for OMSDK dependent iDRAC modules.
- idrac_firmware - This module is enhanced to support proxy.
- idrac_redfish_storage_controller - This module is enhanced to configure controller attributes and online capacity expansion.
- idrac_server_config_profile - This module is enhanced to support proxy settings, import buffer, include in export, and ignore certificate warning.
- idrac_user_info - This module allows to retrieve iDRAC Local user information details.
- ome_domian_user_groups - This module allows to import the LDAP directory groups.
- ome_inventory - This plugin allows to create a inventory from the group on OpenManage Enterprise.

- ome_inventory - This plugin is enhanced to support inventory retrieval of System and Plugin Groups of OpenManage Enterprise.
- ome_profile_info - This module allows to retrieve profiles with attributes on OpenManage Enterprise or OpenManage Enterprise Modular.
- ome_smart_fabric_info - This module retrieves the list of smart fabrics in the inventory of OpenManage Enterprise Modular.
- ome_smart_fabric_uplink_info - This module retrieve details of fabric uplink on OpenManage Enterprise Modular.
- ome_template_network_vlan_info - This module allows to retrieve the network configuration of a template on OpenManage Enterprise or OpenManage Enterprise Modular.

fortinet.fortios

- Add annotations of member operation for every module.
- Support FortiOS v7.0.6, v7.0.7, v7.0.8, v7.2.1, v7.2.2.
- Update `fortios.py` for higher performance;
- supports temporary session key and pre/post login banner;
- update the examples on how to use member operation in Q&A.

junipernetworks.junos

- change gathered key from junos_acls to acls

kubernetes.core

- refactor K8sAnsibleMixin into module_utils/k8s/ (https://github.com/ansible-collections/kubernetes.core/pull/481).

purestorage.fusion

- Patching of resource properties was brought to parity with underlying Python SDK
- fusion_volume - fixed and reorganized, arguments changed

Removed Collections

- dellemc.os10 (previously included version: 1.1.1)
- dellemc.os6 (previously included version: 1.0.7)
- dellemc.os9 (previously included version: 1.0.4)
- mellanox.onyx (previously included version: 1.0.0)

Removed Features

- `dellemc.os10` was considered unmaintained and removed from Ansible 8 as per the removal from Ansible process. Users can still install this collection with `ansible-galaxy collection install dellemc.os10`.
- `dellemc.os6` was considered unmaintained and removed from Ansible 8 as per the removal from Ansible process. Users can still install this collection with `ansible-galaxy collection install dellemc.os6`.
- `dellemc.os9` was considered unmaintained and removed from Ansible 8 as per the removal from Ansible process. Users can still install this collection with `ansible-galaxy collection install dellemc.os9`.
- `mellanox.onyx` was considered unmaintained and removed from Ansible 8 as per the removal from Ansible process. Users can still install this collection with `ansible-galaxy collection install mellanox.onyx`.

Ansible-core

- Remove deprecated `ANSIBLE_CALLBACK_WHITELIST` configuration environment variable, use `ANSIBLE_CALLBACKS_ENABLED` instead. (https://github.com/ansible/ansible/issues/78821)
- Remove deprecated `ANSIBLE_COW_WHITELIST` configuration environment variable, use `ANSIBLE_COW_ACCEPTLIST` instead. (https://github.com/ansible/ansible/issues/78819)
- Remove deprecated `callback_whitelist` configuration option, use `callbacks_enabled` instead. (https://github.com/ansible/ansible/issues/78822)
- Remove deprecated `cow_whitelist` configuration option, use `cowsay_enabled_stencils` instead. (https://github.com/ansible/ansible/issues/78820)

ansible.netcommon

- cli_parse - This plugin was moved to ansible.utils in version 1.0.0, and the redirect to that collection has now been removed.

Deprecated Features

- The cisco.nso collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/155).
- The community.fortios collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/162).
- The community.google collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/160).
- The community.skydive collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/171).

Ansible-core

- The `ConnectionBase()._new_stdin` attribute is deprecated, use `display.prompt_until(msg)` instead.
- ansible-test - The `foreman` test plugin is now deprecated. It will be removed in a future release.
- ansible-test - The `govcsim` simulator in the `vcenter` test plugin is now deprecated. It will be removed in a future release. Users should switch to providing their own test environment through a static configuration file.
- password_hash - deprecate using passlib.hash.hashtype if hashtype isn't in the list of documented choices.
- vars - Specifying a list of dictionaries for `vars:` is deprecated in favor of specifying a dictionary.

amazon.aws

- support for passing both profile and security tokens through a mix of environment variables and parameters has been deprecated and support will be removed in release 6.0.0. After release 6.0.0 it will only be possible to pass either a profile or security tokens, regardless of mechanism used to pass them. To explicitly block a parameter coming from an environment variable pass an empty string as the parameter value. Support for passing profile and security tokens together was originally deprecated in release 1.2.0, however only partially implemented in release 5.0.0 (https://github.com/ansible-collections/amazon.aws/pull/1355).

chocolatey.chocolatey

- win_chocolatey - Deprecate side-by-side installs.

cisco.ios

- ios_bgp_address_family - deprecate neighbors.address/tag/ipv6_adddress with neighbor_address which enables common attributes for facts rendering
- ios_bgp_address_family - deprecate neighbors.password with password_options which allows encryption and password
- ios_bgp_address_family - deprecate slow_peer with slow_peer_options which supports a dict attribute

community.aws

- ecs_service - In a release after 2024-06-01, tha default value of `purge_placement_constraints` will be change from `false` to `true` (https://github.com/ansible-collections/community.aws/pull/1716).
- ecs_service - In a release after 2024-06-01, tha default value of `purge_placement_strategy` will be change from `false` to `true` (https://github.com/ansible-collections/community.aws/pull/1716).
- iam_role - All top level return values other than `iam_role` and `changed` have been deprecated and will be removed in a release after 2023-12-01 (https://github.com/ansible-collections/community.aws/issues/551).
- iam_role - In a release after 2023-12-01 the contents of `assume_role_policy_document` will no longer be converted from CamelCase to snake_case. The `assume_role_policy_document_raw` return value already returns the policy document in this future format (https://github.com/ansible-collections/community.aws/issues/551).

- iam_role_info - In a release after 2023-12-01 the contents of `assume_role_policy_document` will no longer be converted from CamelCase to snake_case. The `assume_role_policy_document_raw` return value already returns the policy document in this future format (https://github.com/ansible-collections/community.aws/issues/551).

community.dns

- The default of the newly added option `txt_character_encoding` will change from `octal` to `decimal` in community.dns 3.0.0. The new default will be compatible with RFC 1035 (https://github.com/ansible-collections/community.dns/pull/134).

community.general

- The `sap` modules `sapcar_extract`, `sap_task_list_execute`, and `hana_query`, will be removed from this collection in community.general 7.0.0 and replaced with redirects to `community.sap_libs`. If you want to continue using these modules, make sure to also install `community.sap_libs` (it is part of the Ansible package) (https://github.com/ansible-collections/community.general/pull/5614).
- consul - deprecate using parameters unused for `state=absent` (https://github.com/ansible-collections/community.general/pull/5772).
- gitlab_runner - the default of the new option `access_level_on_creation` will change from `false` to `true` in community.general 7.0.0. This will cause `access_level` to be used during runner registration as well, and not only during updates (https://github.com/ansible-collections/community.general/pull/5908).
- gitlab_runner - the option `access_level` will lose its default value in community.general 8.0.0. From that version on, you have set this option to `ref_protected` explicitly, if you want to have a protected runner (https://github.com/ansible-collections/community.general/issues/5925).
- manageiq_policies - deprecate `state=list` in favour of using `community.general.manageiq_policies_info` (https://github.com/ansible-collections/community.general/pull/5721).
- rax - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cbs - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cbs_attachments - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cdb - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cdb_database - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cdb_user - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).

- rax_clb - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_clb_nodes - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_clb_ssl - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_dns - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_dns_record - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_facts - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_files - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_files_objects - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_identity - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_keypair - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_meta - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_alarm - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_check - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_entity - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_notification - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).

- rax_mon_notification_plan - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_network - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_queue - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_scaling_group - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_scaling_policy - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).

community.hashi_vault

- ansible-core - support for `ansible-core` versions `2.11` and `2.12` will be dropped in collection version `5.0.0`, making `2.13` the minimum supported version of `ansible-core` (https://github.com/ansible-collections/community.hashi_vault/issues/340).
- hashi_vault lookup - in `v5.0.0` duplicate term string options will raise an exception instead of showing a warning (https://github.com/ansible-collections/community.hashi_vault/issues/356).
- hvac - the minimum version of `hvac` to be supported in collection version `5.0.0` will be at least `1.0.2`; this minimum may be raised before `5.0.0` is released, so please subscribe to the linked issue and look out for new notices in the changelog (https://github.com/ansible-collections/community.hashi_vault/issues/324).

purestorage.fusion

- fusion_hw - hardware module is being removed as changing hardware type has never been supported by Pure Storage Fusion
- fusion_info - nigs subset is deprecated in favor of network_interface_groups and will be removed in the version 1.7.0
- fusion_info - placements subset is deprecated in favor of placement_groups and will be removed in the version 1.7.0
- fusion_pg - placement_engine option is deprecated because Fusion API does not longer support this parameter It will be removed in the version 2.0.0
- fusion_se - parameters ‘addresses’, ‘gateway’ and ‘network_interface_groups’ are deprecated in favor of ‘iscsi’ and will be removed in version 2.0.0
- fusion_tn - tenant networks are being replaced by storage endpoints ``` `fusion_se` ``` and Network Interface Groups ``` `fusion_nig` ```

1.3.2 Ansible 7 Porting Guide

Ansible 7 is based on Ansible-core 2.14.

We suggest you read this page along with the Ansible 7 Changelog to understand what updates you may need to make.

Playbook

- Variables are now evaluated lazily; only when they are actually used. For example, in ansible-core 2.14 an expression `{{ defined_variable or undefined_variable }}` does not fail on `undefined_variable` if the first part of `or` is evaluated to `True` as it is not needed to evaluate the second part. One particular case of a change in behavior to note is the task below which uses the `undefined` test. Prior to version 2.14 this would result in a fatal error trying to access the undefined value in the dictionary. In 2.14 the assertion passes as the dictionary is evaluated as undefined through one of its undefined values:

```
- assert:
    that:
      - some_defined_dict_with_undefined_values is undefined
  vars:
    dict_value: 1
    some_defined_dict_with_undefined_values:
      key1: value1
      key2: '{{ dict_value }}'
      key3: '{{ undefined_dict_value }}'
```

Command Line

- Python 3.9 on the controller node is a hard requirement for this release.
- At startup the filesystem encoding and locale are checked to verify they are UTF-8. If not, the process exits with an error reporting the errant encoding. If you were previously using the `C` or `POSIX` locale, you may be able to use `C.UTF-8`. If you were previously using a locale such as `en_US.ISO-8859-1`, you may be able to use `en_US.UTF-8`. For simplicity it may be easiest to export the appropriate locale using the `LC_ALL` environment variable. An alternative to modifying your system locale is to run Python in UTF-8 mode; See the Python documentation for more information.

Deprecated

No notable changes

Modules

No notable changes

Modules removed

The following modules no longer exist:

- No notable changes

Deprecation notices

No notable changes

Noteworthy module changes

No notable changes

Plugins

No notable changes

Porting custom scripts

No notable changes

Networking

No notable changes

Porting Guide for v7.4.0

Breaking Changes

Ansible-core

- ansible-test - Integration tests which depend on specific file permissions when running in an ansible-test managed host environment may require changes. Tests that require permissions other than `755` or `644` may need to be updated to set the necessary permissions as part of the test run.

Major Changes

community.hrobot

- firewall - Hetzner added output rules support to the firewall. This change unfortunately means that using old versions of the firewall module will always set the output rule list to empty, thus disallowing the server to send out packets (https://github.com/ansible-collections/community.hrobot/issues/75, https://github.com/ansible-collections/community.hrobot/pull/76).

community.vmware

- Use true/false (lowercase) for boolean values in documentation and examples (https://github.com/ansible-collections/community.vmware/issues/1660).

fortinet.fortios

- Add annotations of member operation for every module.
- Update `fortios.py` for higher performance;
- supports temporary session key and pre/post login banner;
- update the examples on how to use member operation in Q&A.

purestorage.fusion

- Patching of resource properties was brought to parity with underlying Python SDK, meaning the collection can create/update/delete all resource properties the SDK can
- fusion_volume - fixed and reorganized, arguments changed

Deprecated Features

amazon.aws

- support for passing both profile and security tokens through a mix of environment variables and parameters has been deprecated and support will be removed in release 6.0.0. After release 6.0.0 it will only be possible to pass either a profile or security tokens, regardless of mechanism used to pass them. To explicitly block a parameter coming from an environment variable pass an empty string as the parameter value. Support for passing profile and security tokens together was originally deprecated in release 1.2.0, however only partially implemented in release 5.0.0 (https://github.com/ansible-collections/amazon.aws/pull/1355).

community.aws

- ecs_service - In a release after 2024-06-01, tha default value of `purge_placement_constraints` will be change from `false` to `true` (https://github.com/ansible-collections/community.aws/pull/1716).
- ecs_service - In a release after 2024-06-01, tha default value of `purge_placement_strategy` will be change from `false` to `true` (https://github.com/ansible-collections/community.aws/pull/1716).
- iam_role - All top level return values other than `iam_role` and `changed` have been deprecated and will be removed in a release after 2023-12-01 (https://github.com/ansible-collections/community.aws/issues/551).
- iam_role - In a release after 2023-12-01 the contents of `assume_role_policy_document` will no longer be converted from CamelCase to snake_case. The `assume_role_policy_document_raw` return value already returns the policy document in this future format (https://github.com/ansible-collections/community.aws/issues/551).
- iam_role_info - In a release after 2023-12-01 the contents of `assume_role_policy_document` will no longer be converted from CamelCase to snake_case. The `assume_role_policy_document_raw` return value already returns the policy document in this future format (https://github.com/ansible-collections/community.aws/issues/551).

community.hashi_vault

- hashi_vault lookup - in `v5.0.0` duplicate term string options will raise an exception instead of showing a warning (https://github.com/ansible-collections/community.hashi_vault/issues/356).

purestorage.fusion

- fusion_hw - hardware module is being removed as changing hardware type has never been supported by Pure Storage Fusion
- fusion_info - nigs subset is deprecated in favor of network_interface_groups and will be removed in the version 1.7.0 (https://github.com/Pure-Storage-Ansible/Fusion-Collection/pull/46).
- fusion_info - placements subset is deprecated in favor of placement_groups and will be removed in the version 1.7.0 (https://github.com/Pure-Storage-Ansible/Fusion-Collection/pull/62).
- fusion_pg - placement_engine option is deprecated because Fusion API does not longer support this parameter It will be removed in the version 2.0.0 (https://github.com/Pure-Storage-Ansible/Fusion-Collection/pull/53).
- fusion_se - parameters “addresses”, “gateway” and “network_interface_groups” are deprecated in favor of “iscsi” and will be removed in version 2.0.0
- fusion_tn - tenant networks are being replaced by storage endpoints `` `fusion_se` `` and Network Interface Groups `` `fusion_nig` ``

Porting Guide for v7.3.0

Breaking Changes

hetzner.hcloud

- inventory plugin - Python v3.5+ is now required.

Major Changes

kubernetes.core

- refactor K8sAnsibleMixin into module_utils/k8s/ (https://github.com/ansible-collections/kubernetes.core/pull/481).

Deprecated Features

- Since the google.cloud collection seems to be maintained again, we cancelled the removal process. So contrary to an earlier announcement, this collection is NOT deprecated and will NOT be removed from Ansible 8 (https://github.com/ansible-community/community-topics/issues/105).

community.general

- gitlab_runner - the option `access_level` will lose its default value in community.general 8.0.0. From that version on, you have set this option to `ref_protected` explicitly, if you want to have a protected runner (https://github.com/ansible-collections/community.general/issues/5925).

Porting Guide for v7.2.0

Added Collections

- dellemc.powerflex (version 1.5.0)
- dellemc.unity (version 1.5.0)

Known Issues

Ansible-core

- ansible-test - Additional configuration may be required for certain container host and container combinations. Further details are available in the testing documentation.
- ansible-test - Custom containers with `VOLUME` instructions may be unable to start, when previously the containers started correctly. Remove the `VOLUME` instructions to resolve the issue. Containers with this condition will cause `ansible-test` to emit a warning.
- ansible-test - Systems with Podman networking issues may be unable to run containers, when previously the issue went unreported. Correct the networking issues to continue using `ansible-test` with Podman.

- ansible-test - Using Docker on systems with SELinux may require setting SELinux to permissive mode. Podman should work with SELinux in enforcing mode.

cisco.meraki

- meraki_network - Updated documentation for *local_status_page_enabled* and *remote_status_page_enabled* as these no longer work.

Breaking Changes

community.general

- ModuleHelper module utils - when the module sets output variables named `msg`, `exception`, `output`, `vars`, or `changed`, the actual output will prefix those names with `_` (underscore symbol) only when they clash with output variables generated by ModuleHelper itself, which only occurs when handling exceptions. Please note that this breaking change does not require a new major release since before this release, it was not possible to add such variables to the output due to a bug (https://github.com/ansible-collections/community.general/pull/5765).

Major Changes

Ansible-core

- ansible-test - Docker Desktop on WSL2 is now supported (additional configuration required).
- ansible-test - Docker and Podman are now supported on hosts with cgroup v2 unified. Previously only cgroup v1 and cgroup v2 hybrid were supported.
- ansible-test - Podman now works on container hosts without systemd. Previously only some containers worked, while others required rootfull or rootless Podman, but would not work with both. Some containers did not work at all.
- ansible-test - Podman on WSL2 is now supported.
- ansible-test - When additional cgroup setup is required on the container host, this will be automatically detected. Instructions on how to configure the host will be provided in the error message shown.

ansible.windows

- Set the minimum Ansible version supported by this collection to Ansible 2.12

chocolatey.chocolatey

- win_chocolatey - Allow users to select the TLS versions used for bootstrapping Chocolatey installation.

Deprecated Features

- The cisco.nso collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/155).
- The community.fortios collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/162).
- The community.google collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/160).
- The community.skydive collection is considered unmaintained and will be removed from Ansible 9 if no one starts maintaining it again before Ansible 9. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/171).

chocolatey.chocolatey

- win_chocolatey - Deprecate side-by-side installs.

cisco.ios

- ios_bgp_address_family - deprecate neighbors.address/tag/ipv6_adddress with neighbor_address which enables common attributes for facts rendering
- ios_bgp_address_family - deprecate neighbors.password with password_options which allows encryption and password
- ios_bgp_address_family - deprecate slow_peer with slow_peer_options which supports a dict attribute

community.dns

- The default of the newly added option `txt_character_encoding` will change from `octal` to `decimal` in community.dns 3.0.0. The new default will be compatible with RFC 1035 (https://github.com/ansible-collections/community.dns/pull/134).

community.general

- consul - deprecate using parameters unused for `state=absent` (https://github.com/ansible-collections/community.general/pull/5772).
- gitlab_runner - the default of the new option `access_level_on_creation` will change from `false` to `true` in community.general 7.0.0. This will cause `access_level` to be used during runner registration as well, and not only during updates (https://github.com/ansible-collections/community.general/pull/5908).
- manageiq_policies - deprecate `state=list` in favour of using `community.general.manageiq_policies_info` (https://github.com/ansible-collections/community.general/pull/5721).
- rax - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).

- rax_cbs - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cbs_attachments - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cdb - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cdb_database - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_cdb_user - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_clb - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_clb_nodes - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_clb_ssl - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_dns - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_dns_record - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_facts - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_files - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_files_objects - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_identity - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_keypair - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).

- rax_meta - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_alarm - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_check - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_entity - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_notification - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_mon_notification_plan - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_network - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_queue - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_scaling_group - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).
- rax_scaling_policy - module relies on deprecates library `pyrax`. Unless maintainers step up to work on the module, it will be marked as deprecated in community.general 7.0.0 and removed in version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5733).

community.hashi_vault

- ansible-core - support for `ansible-core` versions `2.11` and `2.12` will be dropped in collection version `5.0.0`, making `2.13` the minimum supported version of `ansible-core` (https://github.com/ansible-collections/community.hashi_vault/issues/340).
- hvac - the minimum version of `hvac` to be supported in collection version `5.0.0` will be at least `1.0.2`; this minimum may be raised before `5.0.0` is released, so please subscribe to the linked issue and look out for new notices in the changelog (https://github.com/ansible-collections/community.hashi_vault/issues/324).

Porting Guide for v7.1.0

Added Collections

- grafana.grafana (version 1.1.0)

Known Issues

community.routeros

- api_modify - when limits for entries in `queue tree` are defined as human readable - for example `25M` -, the configuration will be correctly set in ROS, but the module will indicate the item is changed on every run even when there was no change done. This is caused by the ROS API which returns the number in bytes - for example `25000000` (which is inconsistent with the CLI behavior). In order to mitigate that, the limits have to be defined in bytes (those will still appear as human readable in the ROS CLI) (https://github.com/ansible-collections/community.routeros/pull/131).
- api_modify, api_info - `routing ospf area`, `routing ospf area range`, `routing ospf instance`, `routing ospf interface-template` paths are not fully implemeted for ROS6 due to the significat changes between ROS6 and ROS7 (https://github.com/ansible-collections/community.routeros/pull/131).

Major Changes

cisco.meraki

- meraki_mr_l7_firewall - New module
- meraki_webhook_payload_template - New module

community.zabbix

- all modules are opting away from zabbix-api and using httpapi ansible.netcommon plugin. We will support zabbix-api for backwards compatibility until next major release. See our README.md for more information about how to migrate
- zabbix_agent and zabbix_proxy roles are opting away from zabbix-api and use httpapi ansible.netcommon plugin. We will support zabbix-api for backwards compatibility until next major release. See our README.md for more information about how to migrate

containers.podman

- New become plugin - podman_unshare
- Podman generate systemd module

fortinet.fortios

- Support FortiOS v7.0.6, v7.0.7, v7.0.8, v7.2.1, v7.2.2.

Deprecated Features

community.general

- The `sap` modules `sapcar_extract`, `sap_task_list_execute`, and `hana_query`, will be removed from this collection in community.general 7.0.0 and replaced with redirects to `community.sap_libs`. If you want to continue using these modules, make sure to also install `community.sap_libs` (it is part of the Ansible package) (https://github.com/ansible-collections/community.general/pull/5614).

Porting Guide for v7.0.0

Added Collections

- ibm.spectrum_virtualize (version 1.10.0)
- inspur.ispim (version 1.2.0)
- lowlydba.sqlserver (version 1.0.4)
- purestorage.fusion (version 1.1.1)
- vultr.cloud (version 1.3.1)

Known Issues

community.routeros

- The `community.routeros.command` module claims to support check mode. Since it cannot judge whether the commands executed modify state or not, this behavior is incorrect. Since this potentially breaks existing playbooks, we will not change this behavior until community.routeros 3.0.0.

dellemc.openmanage

- idrac_user - Issue(192043) The module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_application_alerts_smtp - Issue(212310) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_application_alerts_syslog - Issue(215374) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_device_local_access_configuration - Issue(215035) - The module reports `Successfully updated the local access setting` if an unsupported value is provided for the parameter timeout_limit. However, this value is not actually applied on OpenManage Enterprise Modular.
- ome_device_local_access_configuration - Issue(217865) - The module does not display a proper error message if an unsupported value is provided for the user_defined and lcd_language parameters.

- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_power_settings - Issue(212679) - The module displays the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_device_quick_deploy - Issue(216352) - The module does not display a proper error message if an unsupported value is provided for the ipv6_prefix_length and vlan_id parameters.
- ome_smart_fabric_uplink - Issue(186024) - The module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

netapp.ontap

- na_ontap_snapshot - added documentation to use UTC format for `expiry_time`.

Breaking Changes

- Ansible 7 requires Python 3.9 on the controller, same as ansible-core 2.14.

Ansible-core

- Allow for lazy evaluation of Jinja2 expressions (https://github.com/ansible/ansible/issues/56017)
- The default ansible-galaxy role skeletons no longer contain .travis.yml files. You can configure ansible-galaxy to use a custom role skeleton that contains a .travis.yml file to continue using Galaxy's integration with Travis CI.
- ansible - At startup the filesystem encoding and locale are checked to verify they are UTF-8. If not, the process exits with an error reporting the errant encoding.
- ansible - Increase minimum Python requirement to Python 3.9 for CLI utilities and controller code
- ansible-test - At startup the filesystem encoding is checked to verify it is UTF-8. If not, the process exits with an error reporting the errant encoding.
- ansible-test - At startup the locale is configured as `en_US.UTF-8`, with a fallback to `C.UTF-8`. If neither encoding is available the process exits with an error. If the fallback is used, a warning is displayed. In previous versions the `en_US.UTF-8` locale was always requested. However, no startup checking was performed to verify the locale was successfully configured.
- ansible-test validate-modules - Removed the `missing-python-doc` error code in validate modules, `missing-documentation` is used instead for missing PowerShell module documentation.
- strategy plugins - Make `ignore_unreachable` to increase `ignored` and `ok` and counter, not `skipped` and `unreachable`. (https://github.com/ansible/ansible/issues/77690)

amazon.aws

- Tags beginning with `aws:` will not be removed when purging tags, these tags are reserved by Amazon and may not be updated or deleted (https://github.com/ansible-collections/amazon.aws/issues/817).
- amazon.aws collection - Support for ansible-core < 2.11 has been dropped (https://github.com/ansible-collections/amazon.aws/pull/1087).
- amazon.aws collection - The amazon.aws collection has dropped support for `botocore<1.21.0` and `boto3<1.18.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/amazon.aws/pull/934).
- amazon.aws collection - the `profile` parameter is now mutually exclusive with the `aws_access_key`, `aws_secret_key` and `security_token` parameters (https://github.com/ansible-collections/amazon.aws/pull/834).
- aws_az_info - the module alias `aws_az_facts` was deprecated in Ansible 2.9 and has now been removed (https://github.com/ansible-collections/amazon.aws/pull/832).
- aws_s3 - the default value for `ensure overwrite` has been changed to `different` instead of `always` so that the module is idempotent by default (https://github.com/ansible-collections/amazon.aws/issues/811).
- aws_ssm - on_denied and on_missing now both default to error, for consistency with both aws_secret and the base Lookup class (https://github.com/ansible-collections/amazon.aws/issues/617).
- doc_fragments - remove minimum collection requirements from doc_fragments/aws.py and allow pulling those from doc_fragments/aws_boto3.py instead (https://github.com/ansible-collections/amazon.aws/pull/985).
- ec2 - The `ec2` module has been removed in release 4.0.0 and replaced by the `ec2_instance` module (https://github.com/ansible-collections/amazon.aws/pull/630).
- ec2_ami - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).
- ec2_ami - the parameter aliases `DeviceName`, `VirtualName` and `NoDevice` were previously deprecated and have been removed, please use `device_name`, `virtual_name` and `no_device` instead (https://github.com/ansible-collections/amazon.aws/pull/913).
- ec2_eni_info - the mutual exclusivity of the `eni_id` and `filters` parameters is now enforced, previously `filters` would be ignored if `eni_id` was set (https://github.com/ansible-collections/amazon.aws/pull/954).
- ec2_instance - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).
- ec2_key - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).
- ec2_vol - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).
- ec2_vpc_dhcp_option_info - the parameter aliases `DhcpOptionIds` and `DryRun` were previously deprecated and have been removed, please use `dhcp_options_ids` and `no_device` instead (https://github.com/ansible-collections/amazon.aws/pull/913).
- ec2_vpc_endpoint - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).
- ec2_vpc_igw_info - The default value for `convert_tags` has been changed to `True` (https://github.com/ansible-collections/amazon.aws/pull/835).
- ec2_vpc_net - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).

- ec2_vpc_route_table - the default value for `purge_tags` has been changed from `False` to `True` (https://github.com/ansible-collections/amazon.aws/pull/916).
- elb_classic_lb - the `ec2_elb` fact has been removed (https://github.com/ansible-collections/amazon.aws/pull/827).
- module_utils - Support for the original AWS SDK aka `boto` has been removed, including all relevant helper functions. All modules should now use the `boto3`/`botocore` AWS SDK (https://github.com/ansible-collections/amazon.aws/pull/630)
- s3_bucket - the previously deprecated alias `S3_URL` for the `s3_url` parameter has been removed. Playbooks shuold be updated to use `s3_url` (https://github.com/ansible-collections/amazon.aws/pull/908).
- s3_object - the previously deprecated alias `S3_URL` for the `s3_url` parameter has been removed. Playbooks should be updated to use `s3_url` (https://github.com/ansible-collections/amazon.aws/pull/908).

check_point.mgmt

- cp_mgmt_access_role - the 'machines' parameter now accepts a single str and a new parameter 'machines_list' of type dict has been added. the 'users' parameter now accepts a single str and a new parameter 'users_list' of type dict has been added.
- cp_mgmt_access_rule - the 'vpn' parameter now accepts a single str and a new parameter 'vpn_list' of type dict has been added. the 'position_by_rule' parameter has been changed to 'relative_position' with support of positioning above/below a section (and not just a rule). the 'relative_position' parameter has also 'top' and 'bottom' suboptions which allows positioning a rule at the top and bottom of a section respectively. a new parameter 'search_entire_rulebase' has been added to allow the relative positioning to be unlimited (was previously limited to 50 rules)
- cp_mgmt_administrator - the 'permissions_profile' parameter now accepts a single str and a new parameter 'permissions_profile_list' of type dict has been added.
- cp_mgmt_publish - the 'uid' parameter has been removed.

community.aws

- Tags beginning with `aws:` will not be removed when purging tags, these tags are reserved by Amazon and may not be updated or deleted (https://github.com/ansible-collections/amazon.aws/issues/817).
- acm_certificate - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- autoscaling_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.autoscaling_group`.
- autoscaling_group_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.autoscaling_group_info`.
- aws_secret - tags are no longer removed when the `tags` parameter is not set. To remove all tags set `tags={}` (https://github.com/ansible-collections/community.aws/issues/1146).
- cloudfront_distribution - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- cloudtrail - The module has been migrated to the `amazon.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudtrail`.

- cloudwatch_metric_alarm - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatch_metric_alarm`.
- cloudwatchevent_rule - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchevent_rule`.
- cloudwatchlogs_log_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchlogs_log_group`.
- cloudwatchlogs_log_group_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchlogs_log_group_info`.
- cloudwatchlogs_log_group_metric_filter - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchlogs_log_group_metric_filter`.
- community.aws collection - Support for ansible-core < 2.11 has been dropped (https://github.com/ansible-collections/community.aws/pull/1541).
- community.aws collection - The `community.aws` collection has now dropped support for and any requirements upon the original `boto` AWS SDK, and now uses the `boto3/botocore` AWS SDK (https://github.com/ansible-collections/community.aws/pull/898).
- community.aws collection - The community.aws collection has dropped support for `botocore<1.21.0` and `boto3<1.18.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/community.aws/pull/1362).
- community.aws collection - the `profile` parameter is now mutually exclusive with the `aws_access_key`, `aws_secret_key` and `security_token` parameters (https://github.com/ansible-collections/amazon.aws/pull/834).
- ec2_eip - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_eip`.
- ec2_eip_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_eip_info`.
- ec2_vpc_route_table - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_route_table`.
- ec2_vpc_route_table_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_route_table_info`.
- ec2_vpc_vpn - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- elb_application_lb - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.elb_application_lb`.
- elb_application_lb_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.elb_application_lb_info`.
- elb_instance - the `ec2_elbs` fact has been removed, `updated_elbs` has been added the return values and includes the same information (https://github.com/ansible-collections/community.aws/pull/1173).

- elb_network_lb - the default value of `state` has changed from `absent` to `present` (https://github.com/ansible-collections/community.aws/pull/1167).
- execute_lambda - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.execute_lambda`.
- iam_policy - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_policy`.
- iam_policy_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_policy_info`.
- iam_server_certificate - Passing file names to the `cert`, `chain_cert` and `key` parameters has been removed. We recommend using a lookup plugin to read the files instead, see the documentation for an example (https://github.com/ansible-collections/community.aws/pull/1265).
- iam_server_certificate - the default value for the `dup_ok` parameter has been changed to `true`. To preserve the original behaviour explicitly set the `dup_ok` parameter to `false` (https://github.com/ansible-collections/community.aws/pull/1265).
- iam_user - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_user`.
- iam_user_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_user_info`.
- kms_key - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.kms_key`.
- kms_key - managing the KMS IAM Policy via `policy_mode` and `policy_grant_types` was previously deprecated and has been removed in favor of the `policy` option (https://github.com/ansible-collections/community.aws/pull/1344).
- kms_key - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- kms_key_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.kms_key_info`.
- lambda - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda`.
- lambda_alias - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_alias`.
- lambda_event - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_event`.
- lambda_execute - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_execute`.
- lambda_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_info`.
- lambda_policy - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_policy`.
- rds_cluster - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_cluster`.
- rds_cluster_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_cluster_info`.

- rds_cluster_snapshot - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_cluster_snapshot`.
- rds_instance - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_instance`.
- rds_instance_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_instance_info`.
- rds_instance_snapshot - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_instance_snapshot`.
- rds_option_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_option_group`.
- rds_option_group_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_option_group_info`.
- rds_param_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_param_group`.
- rds_param_group - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- rds_snapshot_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_snapshot_info`.
- rds_subnet_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_subnet_group`.
- route53 - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53`.
- route53_health_check - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53_health_check`.
- route53_health_check - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- route53_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53_info`.
- route53_zone - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53_zone`.
- route53_zone - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).
- script_inventory_ec2 - The ec2.py inventory script has been moved to a new repository. The script can now be downloaded from https://github.com/ansible-community/contrib-scripts/blob/main/inventory/ec2.py and has been removed from this collection. We recommend migrating from the script to the amazon.aws.ec2 inventory plugin. (https://github.com/ansible-collections/community.aws/pull/898)
- sqs_queue - the previously deprecated default value of `purge_tags=False` has been updated to `purge_tags=True` (https://github.com/ansible-collections/community.aws/pull/1343).

community.docker

- This collection does not work with ansible-core 2.11 on Python 3.12+. Please either upgrade to ansible-core 2.12+, or use Python 3.11 or earlier (https://github.com/ansible-collections/community.docker/pull/271).
- docker_container - `exposed_ports` is no longer ignored in `comparisons`. Before, its value was assumed to be identical with the value of `published_ports` (https://github.com/ansible-collections/community.docker/pull/422).
- docker_container - `log_options` can no longer be specified when `log_driver` is not specified (https://github.com/ansible-collections/community.docker/pull/422).
- docker_container - `publish_all_ports` is no longer ignored in `comparisons` (https://github.com/ansible-collections/community.docker/pull/422).
- docker_container - `restart_retries` can no longer be specified when `restart_policy` is not specified (https://github.com/ansible-collections/community.docker/pull/422).
- docker_container - `stop_timeout` is no longer ignored for idempotency if told to be not ignored in `comparisons`. So far it defaulted to `ignore` there, and setting it to `strict` had no effect (https://github.com/ansible-collections/community.docker/pull/422).
- modules and plugins communicating directly with the Docker daemon - when connecting by SSH and not using `use_ssh_client=true`, reject unknown host keys instead of accepting them. This is only a breaking change relative to older community.docker 3.0.0 pre-releases or with respect to Docker SDK for Python < 6.0.0. Docker SDK for Python 6.0.0 will also include this change (https://github.com/ansible-collections/community.docker/pull/434).

community.general

- newrelic_deployment - `revision` is required for v2 API (https://github.com/ansible-collections/community.general/pull/5341).
- scaleway_container_registry_info - no longer replace `secret_environment_variables` in the output by `SENSITIVE_VALUE` (https://github.com/ansible-collections/community.general/pull/5497).

community.hashi_vault

- auth - the default value for `token_validate` has changed from `true` to `false`, as previously announced (https://github.com/ansible-collections/community.hashi_vault/issues/248).
- vault_kv2_get lookup - as previously announced, the default value for `engine_mount_point` in the `vault_kv2_get` lookup has changed from `kv` to `secret` (https://github.com/ansible-collections/community.hashi_vault/issues/279).

community.vmware

- Removed support for ansible-core version < 2.13.0.
- vmware_dvs_portgroup - Add a new sub-option *inherited* to the *in_traffic_shaping* parameter. This means you can keep the setting as-is by not defining the parameter, but also that you have to define the setting as not *inherited* if you want to override it at the PG level (https://github.com/ansible-collections/community.vmware/pull/1483).
- vmware_dvs_portgroup - Add a new sub-option *inherited* to the *out_traffic_shaping* parameter. This means you can keep the setting as-is by not defining the parameter, but also that you have to define the setting as not *inherited* if you want to override it at the PG level (https://github.com/ansible-collections/community.vmware/pull/1483).

- vmware_dvs_portgroup - Change the type of *net_flow* to string to allow setting it implicitly to inherited or to keep the value as-is. This means you can keep the setting as-is by not defining the parameter, but also that while *true* or *no* still work, *True* or *Off* (uppercase) won't (https://github.com/ansible-collections/community.vmware/pull/1483).
- vmware_dvs_portgroup - Remove support for vSphere API less than 6.7.
- vmware_dvs_portgroup - Remove the default for *network_policy* and add a new sub-option *inherited*. This means you can keep the setting as-is by not defining the parameter, but also that you have to define the setting as not *inherited* if you want to override it at the PG level (https://github.com/ansible-collections/community.vmware/pull/1483).
- vmware_dvs_portgroup_info - Remove support for vSphere API less than 6.7.
- vmware_dvswitch - Remove support for vSphere API less than 6.7.
- vmware_dvswitch_uplink_pg - Remove support for vSphere API less than 6.7.
- vmware_guest_boot_manager - Remove default for `secure_boot_enabled` parameter (https://github.com/ansible-collections/community.vmware/issues/1461).
- vmware_vm_config_option - Dict item names in result are changed from strings joined with spaces to strings joined with underlines, e.g. *Guest fullname* is changed to *guest_fullname* (https://github.com/ansible-collections/community.vmware/issues/1268).
- vmware_vspan_session - Remove support for vSphere API less than 6.7.

dellemc.enterprise_sonic

- bgp_af - Add the route_advertise_list dictionary to the argspec to replace the deleted, obsolete advertise_prefix attribute used for SONiC 3.x images on the 1.x branch of this collection. This change corresponds to a SONiC 4.0 OC YANG REST compliance change for the BGP AF REST API. It enables specification of a route map in conjunction with each route advertisement prefix (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/63).
- bgp_af - remove the obsolete 'advertise_prefix' attribute from argspec and config code. This and subsequent co-req replacement with the new route advertise list argument structure require corresponding changes in playbooks previoulsly used for configuring route advertise prefixes for SONiC 3.x images. (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/60)
- bgp_neighbors - Replace the previously defined standalone "bfd" attribute with a bfd dictionary containing multiple attributes. This change corresponds to the revised SONiC 4.x implementation of OC YANG compatible REST APIs. Playbooks previously using the bfd attributes for SONiC 3.x images must be modified for useon SONiC 4.0 images to use the new definition for the bfd attribute argspec structure (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/72).
- bgp_neighbors - Replace, for BGP peer groups, the previously defined standalone "bfd" attribute with a bfd dictionary containing multiple attributes. This change corresponds to the revised SONiC 4.x implementation of OC YANG compatible REST APIs. Playbooks previously using the bfd attributes for SONiC 3.x images must be modified for useon SONiC 4.0 images to use the new definition for the bfd attribute argspec structure (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/81).

Major Changes

Ansible-core

- Move handler processing into new `PlayIterator` phase to use the configured strategy (https://github.com/ansible/ansible/issues/65067)
- ansible - At startup the filesystem encoding and locale are checked to verify they are UTF-8. If not, the process exits with an error reporting the errant encoding.
- ansible - Increase minimum Python requirement to Python 3.9 for CLI utilities and controller code
- ansible-test - At startup the filesystem encoding is checked to verify it is UTF-8. If not, the process exits with an error reporting the errant encoding.
- ansible-test - At startup the locale is configured as `en_US.UTF-8`, with a fallback to `C.UTF-8`. If neither encoding is available the process exits with an error. If the fallback is used, a warning is displayed. In previous versions the `en_US.UTF-8` locale was always requested. However, no startup checking was performed to verify the locale was successfully configured.

amazon.aws

- amazon.aws collection - The amazon.aws collection has dropped support for `botocore<1.20.0` and `boto3<1.17.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/amazon.aws/pull/574).
- autoscaling_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.autoscaling_group`.
- autoscaling_group_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.autoscaling_group_info`.
- cloudtrail - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudtrail`.
- cloudwatch_metric_alarm - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatch_metric_alarm`.
- cloudwatchevent_rule - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchevent_rule`.
- cloudwatchlogs_log_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchlogs_log_group`.
- cloudwatchlogs_log_group_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchlogs_log_group_info`.
- cloudwatchlogs_log_group_metric_filter - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.cloudwatchlogs_log_group_metric_filter`.
- ec2_eip - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_eip`.

- ec2_eip_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_eip_info`.
- elb_application_lb - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.elb_application_lb`.
- elb_application_lb_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.elb_application_lb_info`.
- execute_lambda - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.execute_lambda`.
- iam_policy - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_policy`.
- iam_policy_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_policy_info`.
- iam_user - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_user`.
- iam_user_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.iam_user_info`.
- kms_key - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.kms_key`.
- kms_key_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.kms_key_info`.
- lambda - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda`.
- lambda_alias - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_alias`.
- lambda_event - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_event`.
- lambda_execute - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_execute`.
- lambda_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_info`.
- lambda_policy - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.lambda_policy`.
- rds_cluster - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_cluster`.
- rds_cluster_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_cluster_info`.
- rds_cluster_snapshot - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_cluster_snapshot`.
- rds_instance - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_instance`.
- rds_instance_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_instance_info`.

- rds_instance_snapshot - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_instance_snapshot`.
- rds_option_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_option_group`.
- rds_option_group_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_option_group_info`.
- rds_param_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_param_group`.
- rds_snapshot_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_snapshot_info`.
- rds_subnet_group - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.rds_subnet_group`.
- route53 - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53`.
- route53_health_check - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53_health_check`.
- route53_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53_info`.
- route53_zone - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.route53_zone`.

arista.eos

- Remove following EOS dprecated modules
- Use of connection: local and the provider option are no longer valid on any modules in this collection.
- eos_interface
- eos_l2_interface
- eos_l3_interface
- eos_linkagg
- eos_static_route
- eos_vlan

check_point.mgmt

- plugins/httpapi/checkpoint - Support for Smart-1 Cloud with new variable ‘ansible_cloud_mgmt_id’

chocolatey.chocolatey

- win_chocolatey - Added bootstrap_script option to allow users to target a script URL for installing Chocolatey on clients.
- win_chocolatey_facts - Added outdated packages list to data returned.

cisco.asa

- Please use either of the following connection types - network_cli, httpapi or netconf.
- This includes the following modules:
- This release drops support for *connection: local* and provider dictionary.
- This release removes all deprecated plugins that have reached their end-of-life.
- Use of connection: local and the provider option are no longer valid on any modules in this collection.
- asa_acl
- asa_og

cisco.ios

- Only valid connection types for this collection is network_cli.
- This release drops support for *connection: local* and provider dictionary.

cisco.iosxr

- Only valid connection types for this collection are network_cli and netconf.
- This release drops support for *connection: local* and provider dictionary.

cisco.nxos

- Please use either of the following connection types - network_cli, httpapi or netconf.
- This release drops support for *connection: local* and provider dictionary.

community.aws

- community.aws collection - The amazon.aws collection has dropped support for `botocore<1.20.0` and `boto3<1.17.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/community.aws/pull/956).

community.docker

- The collection now contains vendored code from the Docker SDK for Python to talk to the Docker daemon. Modules and plugins using this code no longer need the Docker SDK for Python installed on the machine the module or plugin is running on (https://github.com/ansible-collections/community.docker/pull/398).
- docker_api connection plugin - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/414).
- docker_container - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/422).
- docker_container - the module was completely rewritten from scratch (https://github.com/ansible-collections/community.docker/pull/422).
- docker_container_exec - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/401).
- docker_container_info - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/402).
- docker_containers inventory plugin - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/413).
- docker_host_info - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/403).
- docker_image - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/404).
- docker_image_info - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/405).
- docker_image_load - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/406).
- docker_login - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/407).
- docker_network - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/408).

- docker_network_info - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/409).
- docker_plugin - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/429).
- docker_prune - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/410).
- docker_volume - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/411).
- docker_volume_info - no longer uses the Docker SDK for Python. It requires `requests` to be installed, and depending on the features used has some more requirements. If the Docker SDK for Python is installed, these requirements are likely met (https://github.com/ansible-collections/community.docker/pull/412).

community.general

- The internal structure of the collection was changed for modules and action plugins. These no longer live in a directory hierarchy ordered by topic, but instead are now all in a single (flat) directory. This has no impact on users *assuming they did not use internal FQCNs*. These will still work, but result in deprecation warnings. They were never officially supported and thus the redirects are kept as a courtsey, and this is not labelled as a breaking change. Note that for example the Ansible VScode plugin started recommending these internal names. If you followed its recommendation, you will now have to change back to the short names to avoid deprecation warnings, and potential errors in the future as these redirects will be removed in community.general 9.0.0 (https://github.com/ansible-collections/community.general/pull/5461).
- newrelic_deployment - removed New Relic v1 API, added support for v2 API (https://github.com/ansible-collections/community.general/pull/5341).

community.mysql

- mysql_db - the `pipefail` argument's default value will be changed to `true` in community.mysql 4.0.0. If your target machines do not use `bash` as a default interpreter, set `pipefail` to `false` explicitly. However, we strongly recommend setting up `bash` as a default and `pipefail=true` as it will protect you from getting broken dumps you don't know about (https://github.com/ansible-collections/community.mysql/issues/407).

community.network

- The community.network collection no longer supports Ansible 2.9 and ansible-base 2.10. While we take no active measures to prevent usage, we will remove compatibility code and other compatility measures that will effectively prevent using most content from this collection with Ansible 2.9, and some content of this collection with ansible-base 2.10. Both Ansible 2.9 and ansible-base 2.10 will very soon be End of Life and if you are still using them, you should consider upgrading to ansible-core 2.11 or later as soon as possible (https://github.com/ansible-collections/community.network/pull/426).
- The internal structure of the collection was changed for modules and action plugins. These no longer live in a directory hierarchy ordered by topic, but instead are now all in a single (flat) directory. This has no impact on users *assuming they did not use internal FQCNs*. These will still work, but result in deprecation warnings. They were never officially supported and thus the redirects are kept as a courtsey, and this is not labelled as a

breaking change. Note that for example the Ansible VScode plugin started recommending these internal names. If you followed its recommendation, you will now have to change back to the short names to avoid deprecation warnings, and potential errors in the future as these redirects will be removed in community.network 8.0.0 (https://github.com/ansible-collections/community.network/pull/482).

community.postgresql

- postgresql_user - the `groups` argument has been deprecated and will be removed in `community.postgresql 3.0.0`. Please use the `postgresql_membership` module to specify group/role memberships instead (https://github.com/ansible-collections/community.postgresql/issues/277).

dellemc.enterprise_sonic

- Added 'static_routes' module to collection (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/82).
- Added a resource module for NTP support (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/99).
- Added a resource module for support of prefix lists (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/100).
- Updated backend REST API request formats in all applicable modules for compatibility with SONiC 4.x openconfig YANG compliant REST APIs. (https://github.com/ansible-collections/dellemc.enterprise_sonic/pull/53)

dellemc.openmanage

- Added collection metadata for creating execution environments.
- Refactored the Markdown (MD) files and content for better readability.
- The share parameters are deprecated from the following modules - idrac_network, idrac_timezone_ntp, dellemc_configure_idrac_eventing, dellemc_configure_idrac_services, dellemc_idrac_lc_attributes, dellemc_system_lockdown_mode.
- idrac_bios - The module is enhanced to support clear pending BIOS attributes, reset BIOS to default settings, and configure BIOS attribute using Redfish.
- idrac_boot - Support for configuring the boot settings on iDRAC.
- idrac_redfish_storage_controller - This module is enhanced to support LockVirtualDisk operation.
- idrac_virtual_media - This module allows to configure Remote File Share settings.
- ome_device_group - The module is enhanced to support the removal of devices from a static device group.
- ome_devices - Support for performing device-specific operations on OpenManage Enterprise.

fortinet.fortimanager

- Fix compatibility issue for ansible 2.9.x and ansible-base 2.10.x.
- Many fixes for Ansible sanity test warnings & errors.
- Support FortiManager Schema 7.2.0 , 98 new modules
- support Ansible changelogs.

fortinet.fortios

- Support Diff feature in check_mode.
- Support Fortios 7.2.0.

infoblox.nios_modules

- Feature for extra layer security , with *cert* and *key* parameters in playbooks for authenticating using certificate and key `*.pem` file absolute path #154
- Fix to remove issue causing due to template attr in deleting network using Ansible module nios network #147
- Update *text* field of TXT Record #128
- Update operation using *old_name* and *new_name* for the object with dummy name in *old_name* (which does not exist in system) will not create a new object in the system. An error will be thrown stating the object does not exist in the system #129

junipernetworks.junos

- Use of connection: local and the provider option are no longer valid on any modules in this collection.

vyos.vyos

- Use of connection: local and the provider option are no longer valid on any modules in this collection.

Removed Collections

- servicenow.servicenow (previously included version: 1.0.6)

Removed Features

Ansible-core

- PlayIterator - remove deprecated `PlayIterator.ITERATING_*` and `PlayIterator.FAILED_*`
- Remove deprecated `ALLOW_WORLD_READABLE_TMPFILES` configuration option (https://github.com/ansible/ansible/issues/77393)
- Remove deprecated `COMMAND_WARNINGS` configuration option (https://github.com/ansible/ansible/issues/77394)

- Remove deprecated `DISPLAY_SKIPPED_HOSTS` environment variable (https://github.com/ansible/ansible/issues/77396)
- Remove deprecated `LIBVIRT_LXC_NOSECLABEL` environment variable (https://github.com/ansible/ansible/issues/77395)
- Remove deprecated `NETWORK_GROUP_MODULES` environment variable (https://github.com/ansible/ansible/issues/77397)
- Remove deprecated `UnsafeProxy`
- Remove deprecated `plugin_filters_cfg` config option from `default` section (https://github.com/ansible/ansible/issues/77398)
- Remove deprecated functionality that allows loading cache plugins directly without using `cache_loader`.
- Remove deprecated functionality that allows subclassing `DefaultCallback` without the corresponding `doc_fragment`.
- Remove deprecated powershell functions `Load-CommandUtils` and `Import-PrivilegeUtil`
- apt_key - remove deprecated `key` module param
- command/shell - remove deprecated `warn` module param
- get_url - remove deprecated `sha256sum` module param
- import_playbook - remove deprecated functionality that allows providing additional parameters in free form

amazon.aws

- cloudformation - the `template_format` option has been removed. It has been ignored by the module since Ansible 2.3 (https://github.com/ansible-collections/amazon.aws/pull/833).
- ec2_key - the `wait_timeout` option had no effect, was deprecated in release 1.0.0, and has now been removed (https://github.com/ansible-collections/amazon.aws/pull/830).
- ec2_key - the `wait` option had no effect, was deprecated in release 1.0.0, and has now been removed (https://github.com/ansible-collections/amazon.aws/pull/830).
- ec2_tag - the previously deprecated state `list` has been removed. To list tags on an EC2 resource the `ec2_tag_info` module can be used (https://github.com/ansible-collections/amazon.aws/pull/829).
- ec2_vol - the previously deprecated state `list` has been removed. To list volumes the `ec2_vol_info` module can be used (https://github.com/ansible-collections/amazon.aws/pull/828).
- module_utils.batch - the class `ansible_collections.amazon.aws.plugins.module_utils.batch.AWSConnection` has been removed. Please use `AnsibleAWSModule.client()` instead (https://github.com/ansible-collections/amazon.aws/pull/831).

ansible.netcommon

- napalm - Removed unused connection plugin.
- net_banner - Use <network_os>_banner instead.
- net_interface - Use <network_os>_interfaces instead.
- net_l2_interface - Use <network_os>_l2_interfaces instead.
- net_l3_interface - Use <network_os>_l3_interfaces instead.

- net_linkagg - Use <network_os>_lag_interfaces instead.
- net_lldp - Use <network_os>_lldp_global instead.
- net_lldp_interface - Use <network_os>_lldp_interfaces instead.
- net_logging - Use <network_os>_logging_global instead.
- net_static_route - Use <network_os>_static_routes instead.
- net_system - Use <network_os>_system instead.
- net_user - Use <network_os>_user instead.
- net_vlan - Use <network_os>_vlans instead.
- net_vrf - Use <network_os>_vrf instead.

cisco.ios

- ios_interface - use ios_interfaces instead.
- ios_l2_interface - use ios_l2_interfaces instead.
- ios_l3_interface - use ios_l3_interfaces instead.
- ios_static_route - use ios_static_routes instead.
- ios_vlan - use ios_vlans instead.

cisco.iosxr

- iosxr_interface - use iosxr_interfaces instead.

cisco.nxos

- This release removes the following deprecated plugins that have reached their end-of-life.
- nxos_acl
- nxos_acl_interface
- nxos_interface
- nxos_interface_ospf
- nxos_l2_interface
- nxos_l3_interface
- nxos_linkagg
- nxos_lldp
- nxos_ospf
- nxos_ospf_vrf
- nxos_smu
- nxos_static_route
- nxos_vlan

community.aws

- aws_kms_info - the unused and deprecated `keys_attr` parameter has been removed (https://github.com/ansible-collections/amazon.aws/pull/1172).
- data_pipeline - the `version` option has always been ignored and has been removed (https://github.com/ansible-collections/community.aws/pull/1160"
- ec2_eip - The `wait_timeout` option has been removed. It has always been ignored by the module (https://github.com/ansible-collections/community.aws/pull/1159).
- ec2_lc - the `associate_public_ip_address` option has been removed. It has always been ignored by the module (https://github.com/ansible-collections/community.aws/pull/1158).
- ec2_metric_alarm - support for using the <=, <, > and >= operators for comparison has been dropped. Please use `LessThanOrEqualToThreshold`, `LessThanThreshold`, `GreaterThanThreshold` or `GreaterThanOrEqualToThreshold` instead (https://github.com/ansible-collections/amazon.aws/pull/1164).
- ecs_ecr - The deprecated alias `delete_policy` has been removed. Please use `purge_policy` instead (https://github.com/ansible-collections/community.aws/pull/1161).
- iam_managed_policy - the unused `fail_on_delete` parameter has been removed (https://github.com/ansible-collections/community.aws/pull/1168)
- s3_lifecycle - the unused parameter `requester_pays` has been removed (https://github.com/ansible-collections/community.aws/pull/1165).
- s3_sync - remove unused `retries` parameter (https://github.com/ansible-collections/community.aws/pull/1166).

community.azure

- azure_rm_aks_facts, azure_rm_aks_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_aks_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_aksversion_facts, azure_rm_aksversion_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_aksversion_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_applicationsecuritygroup_facts, azure_rm_applicationsecuritygroup_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_applicationsecuritygroup_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_appserviceplan_facts, azure_rm_appserviceplan_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_appserviceplan_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_automationaccount_facts, azure_rm_automationaccount_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_automationaccount_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_autoscale_facts, azure_rm_autoscale_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_autoscale_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_availabilityset_facts, azure_rm_availabilityset_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_availabilityset_info instead (https://github.com/ansible-collections/community.azure/pull/31).

- azure_rm_cdnendpoint_facts, azure_rm_cdnendpoint_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_cdnendpoint_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_cdnprofile_facts, azure_rm_cdnprofile_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_cdnprofile_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_containerinstance_facts, azure_rm_containerinstance_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_containerinstance_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_containerregistry_facts, azure_rm_containerregistry_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_containerregistry_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_cosmosdbaccount_facts, azure_rm_cosmosdbaccount_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_cosmosdbaccount_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_deployment_facts, azure_rm_deployment_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_deployment_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlab_facts, azure_rm_devtestlab_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlab_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabarmtemplate_facts, azure_rm_devtestlabarmtemplate_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabarmtemplate_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabartifact_facts, azure_rm_devtestlabartifact_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabartifact_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabartifactsource_facts, azure_rm_devtestlabartifactsource_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabartifactsource_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabcustomimage_facts, azure_rm_devtestlabcustomimage_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabcustomimage_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabenvironment_facts, azure_rm_devtestlabenvironment_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabenvironment_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabpolicy_facts, azure_rm_devtestlabpolicy_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabpolicy_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabschedule_facts, azure_rm_devtestlabschedule_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabschedule_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_devtestlabvirtualmachine_facts, azure_rm_devtestlabvirtualmachine_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabvirtualmachine_info instead (https://github.com/ansible-collections/community.azure/pull/31).

- azure_rm_devtestlabvirtualnetwork_facts, azure_rm_devtestlabvirtualnetwork_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_devtestlabvirtualnetwork_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_dnsrecordset_facts, azure_rm_dnsrecordset_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_dnsrecordset_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_dnszone_facts, azure_rm_dnszone_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_dnszone_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_functionapp_facts, azure_rm_functionapp_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_functionapp_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_hdinsightcluster_facts, azure_rm_hdinsightcluster_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_hdinsightcluster_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_image_facts, azure_rm_image_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_image_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_loadbalancer_facts, azure_rm_loadbalancer_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_loadbalancer_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_lock_facts, azure_rm_lock_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_lock_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_loganalyticsworkspace_facts, azure_rm_loganalyticsworkspace_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_loganalyticsworkspace_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_managed_disk, azure_rm_manageddisk - the deprecated modules have been removed. Use azure.azcollection.azure_rm_manageddisk instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_managed_disk_facts, azure_rm_manageddisk_facts, azure_rm_manageddisk_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_manageddisk_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mariadbconfiguration_facts, azure_rm_mariadbconfiguration_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mariadbconfiguration_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mariadbdatabase_facts, azure_rm_mariadbdatabase_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mariadbdatabase_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mariadbfirewallrule_facts, azure_rm_mariadbfirewallrule_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mariadbfirewallrule_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mariadbserver_facts, azure_rm_mariadbserver_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mariadbserver_info instead (https://github.com/ansible-collections/community.azure/pull/31).

- azure_rm_mysqlconfiguration_facts, azure_rm_mysqlconfiguration_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mysqlconfiguration_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mysqldatabase_facts, azure_rm_mysqldatabase_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mysqldatabase_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mysqlfirewallrule_facts, azure_rm_mysqlfirewallrule_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mysqlfirewallrule_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_mysqlserver_facts, azure_rm_mysqlserver_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_mysqlserver_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_networkinterface_facts, azure_rm_networkinterface_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_networkinterface_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_postgresqlconfiguration_facts, azure_rm_postgresqlconfiguration_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_postgresqlconfiguration_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_postgresqldatabase_facts, azure_rm_postgresqldatabase_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_postgresqldatabase_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_postgresqlfirewallrule_facts, azure_rm_postgresqlfirewallrule_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_postgresqlfirewallrule_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_postgresqlserver_facts, azure_rm_postgresqlserver_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_postgresqlserver_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_publicipaddress_facts, azure_rm_publicipaddress_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_publicipaddress_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_rediscache_facts, azure_rm_rediscache_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_rediscache_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_resource_facts, azure_rm_resource_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_resource_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_resourcegroup_facts, azure_rm_resourcegroup_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_resourcegroup_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_roleassignment_facts, azure_rm_roleassignment_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_roleassignment_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_roledefinition_facts, azure_rm_roledefinition_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_roledefinition_info instead (https://github.com/ansible-collections/community.azure/pull/31).

- azure_rm_routetable_facts, azure_rm_routetable_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_routetable_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_securitygroup_facts, azure_rm_securitygroup_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_securitygroup_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_servicebus_facts, azure_rm_servicebus_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_servicebus_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_sqldatabase_facts, azure_rm_sqldatabase_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_sqldatabase_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_sqlfirewallrule_facts, azure_rm_sqlfirewallrule_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_sqlfirewallrule_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_sqlserver_facts, azure_rm_sqlserver_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_sqlserver_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_storageaccount_facts, azure_rm_storageaccount_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_storageaccount_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_subnet_facts, azure_rm_subnet_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_subnet_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_trafficmanagerendpoint_facts, azure_rm_trafficmanagerendpoint_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_trafficmanagerendpoint_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_trafficmanagerprofile_facts, azure_rm_trafficmanagerprofile_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_trafficmanagerprofile_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachine_extension, azure_rm_virtualmachineextension - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachineextension instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachine_facts, azure_rm_virtualmachine_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachine_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachine_scaleset, azure_rm_virtualmachinescaleset - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachinescaleset instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachine_scaleset_facts, azure_rm_virtualmachinescaleset_facts, azure_rm_virtualmachinescaleset_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachinescaleset_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachineextension_facts, azure_rm_virtualmachineextension_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachineextension_info instead (https://github.com/ansible-collections/community.azure/pull/31).

- azure_rm_virtualmachineimage_facts, azure_rm_virtualmachineimage_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachineimage_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachinescalesetextension_facts, azure_rm_virtualmachinescalesetextension_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachinescalesetextension_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualmachinescalesetinstance_facts, azure_rm_virtualmachinescalesetinstance_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualmachinescalesetinstance_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualnetwork_facts, azure_rm_virtualnetwork_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualnetwork_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_virtualnetworkpeering_facts, azure_rm_virtualnetworkpeering_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_virtualnetworkpeering_info instead (https://github.com/ansible-collections/community.azure/pull/31).
- azure_rm_webapp_facts, azure_rm_webapp_info - the deprecated modules have been removed. Use azure.azcollection.azure_rm_webapp_info instead (https://github.com/ansible-collections/community.azure/pull/31).

community.docker

- Execution Environments built with community.docker no longer include docker-compose < 2.0.0. If you need to use it with the `docker_compose` module, please install that requirement manually (https://github.com/ansible-collections/community.docker/pull/400).
- Support for Ansible 2.9 and ansible-base 2.10 has been removed. If you need support for Ansible 2.9 or ansible-base 2.10, please use community.docker 2.x.y (https://github.com/ansible-collections/community.docker/pull/400).
- Support for Docker API versions 1.20 to 1.24 has been removed. If you need support for these API versions, please use community.docker 2.x.y (https://github.com/ansible-collections/community.docker/pull/400).
- Support for Python 2.6 has been removed. If you need support for Python 2.6, please use community.docker 2.x.y (https://github.com/ansible-collections/community.docker/pull/400).
- Various modules - the default of `tls_hostname` (`localhost`) has been removed. If you want to continue using `localhost`, you need to specify it explicitly (https://github.com/ansible-collections/community.docker/pull/363).
- docker_container - the `all` value is no longer allowed in `published_ports`. Use `publish_all_ports=true` instead (https://github.com/ansible-collections/community.docker/pull/399).
- docker_container - the default of `command_handling` was changed from `compatibility` to `correct`. Older versions were warning for every invocation of the module when this would result in a change of behavior (https://github.com/ansible-collections/community.docker/pull/399).
- docker_stack - the return values `out` and `err` have been removed. Use `stdout` and `stderr` instead (https://github.com/ansible-collections/community.docker/pull/363).

community.general

- bitbucket* modules - `username` is no longer an alias of `workspace`, but of `user` (https://github.com/ansible-collections/community.general/pull/5326).
- gem - the default of the `norc` option changed from `false` to `true` (https://github.com/ansible-collections/community.general/pull/5326).
- gitlab_group_members - `gitlab_group` must now always contain the full path, and no longer just the name or path (https://github.com/ansible-collections/community.general/pull/5326).
- keycloak_authentication - the return value `flow` has been removed. Use `end_state` instead (https://github.com/ansible-collections/community.general/pull/5326).
- keycloak_group - the return value `group` has been removed. Use `end_state` instead (https://github.com/ansible-collections/community.general/pull/5326).
- lxd_container - the default of the `ignore_volatile_options` option changed from `true` to `false` (https://github.com/ansible-collections/community.general/pull/5326).
- mail callback plugin - the `sender` option is now required (https://github.com/ansible-collections/community.general/pull/5326).
- module_helper module utils - remove the `VarDict` attribute from `ModuleHelper`. Import `VarDict` from `ansible_collections.community.general.plugins.module_utils.mh.mixins.vars` instead (https://github.com/ansible-collections/community.general/pull/5326).
- proxmox inventory plugin - the default of the `want_proxmox_nodes_ansible_host` option changed from `true` to `false` (https://github.com/ansible-collections/community.general/pull/5326).
- vmadm - the `debug` option has been removed. It was not used anyway (https://github.com/ansible-collections/community.general/pull/5326).

community.network

- aireos modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- aireos modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- aruba modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- aruba modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ce modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ce modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- enos modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- enos modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ironware modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).

- ironware modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- sros modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- sros modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).

community.vmware

- vca_fw - The deprecated module `vca_fw` has been removed.
- vca_nat - The deprecated module `vca_nat` has been removed.
- vca_vapp - The deprecated module `vca_vapp` has been removed.
- vmware_dns_config - The deprecated module `vmware_dns_config` has been removed, you can use `vmware_host_dns` instead.
- vmware_guest_network - The deprecated parameter `networks` has been removed, use loops to handle multiple interfaces (https://github.com/ansible-collections/community.vmware/pull/1459).
- vmware_guest_vnc - The deprecated module `vmware_guest_vnc` has been removed. The VNC support has been dropped with vSphere 7 and later (https://github.com/ansible-collections/community.vmware/pull/1454).
- vmware_host_firewall_manager - The module doesn't accept a list for `allowed_hosts` anymore, use a dict instead. Additionally, `all_ip` is now a required sub-option of `allowed_hosts` (https://github.com/ansible-collections/community.vmware/pull/1463).
- vsphere_copy - The deprecated parameters `host` and `login` have been removed. Use `hostname` and `username` instead (https://github.com/ansible-collections/community.vmware/pull/1456).

junipernetworks.junos

- Remove following deprecated Junos Modules.
- junos_interface
- junos_l2_interface
- junos_l3_interface
- junos_linkagg
- junos_lldp
- junos_lldp_interface
- junos_static_route
- junos_vlan

vyos.vyos

- vyos_interface - use vyos_interfaces instead.
- vyos_l3_interface - use vyos_l3_interfaces instead.
- vyos_linkagg - use vyos_lag_interfaces instead.
- vyos_lldp - use vyos_lldp_global instead.
- vyos_lldp_interface - use vyos_lldp_interfaces instead.
- vyos_static_route - use vyos_static_routes instead.

Deprecated Features

- The dellemc.os10 collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/134).
- The dellemc.os6 collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/132).
- The dellemc.os9 collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/133).
- The google.cloud collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/105).
- The mellanox.onyx collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/136).

Ansible-core

- Deprecate ability of lookup plugins to return arbitrary data. Lookup plugins must return lists, failing to do so will be an error in 2.18. (https://github.com/ansible/ansible/issues/77788)
- Encryption - Deprecate use of the Python crypt module due to it's impending removal from Python 3.13
- PlayContext.verbosity is deprecated and will be removed in 2.18. Use ansible.utils.display.Display().verbosity as the single source of truth.
- `DEFAULT_FACT_PATH`, `DEFAULT_GATHER_SUBSET` and `DEFAULT_GATHER_TIMEOUT` are deprecated and will be removed in 2.18. Use `module_defaults` keyword instead.
- `PlayIterator` - deprecate `cache_block_tasks` and `get_original_task` which are noop and unused.
- `Templar` - deprecate `shared_loader_obj` option which is unused. `ansible.plugins.loader` is used directly instead.
- listify_lookup_plugin_terms, deprecate 'loader/dataloader' parameter as it not used.
- vars plugins - determining whether or not to run ansible.legacy vars plugins with the class attribute REQUIRES_WHITELIST is deprecated, set REQUIRES_ENABLED instead.

amazon.aws

- amazon.aws collection - Support for the `EC2_ACCESS_KEY` environment variable has been deprecated and will be removed in a release after 2024-12-01. Please use the `access_key` parameter or `AWS_ACCESS_KEY_ID` environment variable instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - Support for the `EC2_REGION` environment variable has been deprecated and will be removed in a release after 2024-12-01. Please use the `region` parameter or `AWS_REGION` environment variable instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - Support for the `EC2_SECRET_KEY` environment variable has been deprecated and will be removed in a release after 2024-12-01. Please use the `secret_key` parameter or `AWS_SECRET_ACCESS_KEY` environment variable instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - Support for the `EC2_SECURITY_TOKEN` environment variable has been deprecated and will be removed in a release after 2024-12-01. Please use the `session_token` parameter or `AWS_SESSION_TOKEN` environment variable instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - Support for the `EC2_URL` and `S3_URL` environment variables has been deprecated and will be removed in a release after 2024-12-01. Please use the `endpoint_url` parameter or `AWS_ENDPOINT_URL` environment variable instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - The `access_token` alias for the `session_token` parameter has been deprecated and will be removed in a release after 2024-12-01. Please use the `session_token` name instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - The `aws_security_token` alias for the `session_token` parameter has been deprecated and will be removed in a release after 2024-12-01. Please use the `session_token` name instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - The `ec2_access_key` alias for the `access_key` parameter has been deprecated and will be removed in a release after 2024-12-01. Please use the `access_key` name instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - The `ec2_region` alias for the `region` parameter has been deprecated and will be removed in a release after 2024-12-01. Please use the `region` name instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - The `ec2_secret_key` alias for the `secret_key` parameter has been deprecated and will be removed in a release after 2024-12-01. Please use the `secret_key` name instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - The `security_token` alias for the `session_token` parameter has been deprecated and will be removed in a release after 2024-12-01. Please use the `session_token` name instead (https://github.com/ansible-collections/amazon.aws/pull/1172).
- amazon.aws collection - due to the AWS SDKs announcing the end of support for Python less than 3.7 (https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/) support for Python less than 3.7 by this collection has been deprecated and will be removed in a release after 2023-05-31 (https://github.com/ansible-collections/amazon.aws/pull/935).
- aws_s3 - The `S3_URL` alias for the s3_url option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).
- ec2_ami - The `DeviceName` alias for the device_name option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).
- ec2_ami - The `NoDevice` alias for the no_device option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).

- ec2_ami - The VirtualName alias for the virtual_name option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).
- ec2_ami - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/846).
- ec2_instance - The default value for `instance_type` has been deprecated, in the future release you must set an instance_type or a launch_template (https://github.com/ansible-collections/amazon.aws/pull/587).
- ec2_instance - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/849).
- ec2_key - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/846).
- ec2_security_group - support for passing nested lists to cidr_ip and cidr_ipv6 has been deprecated. Nested lists can be passed through the flatten filter instead cidr_ip: '{{ my_cidrs | flatten }}' (https://github.com/ansible-collections/amazon.aws/pull/1213).
- ec2_vol - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/846).
- ec2_vpc_dhcp_option_info - The DhcpOptionIds alias for the dhcp_option_ids option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).
- ec2_vpc_dhcp_option_info - The DryRun alias for the dry_run option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).
- ec2_vpc_endpoint - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/846).
- ec2_vpc_net - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/848).
- ec2_vpc_route_table - the current default value of False for purge_tags has been deprecated and will be updated in release 5.0.0 to True (https://github.com/ansible-collections/amazon.aws/pull/846).
- inventory/aws_ec2 - the include_extra_api_calls is now deprecated, its value is silently ignored (https://github.com/ansible-collections/amazon.aws/pull/1097).
- module_utils.cloud - removal of the CloudRetry.backoff has been delayed until release 6.0.0. It is recommended to update custom modules to use jittered_backoff or exponential_backoff instead (https://github.com/ansible-collections/amazon.aws/pull/951).
- module_utils.url - ansible_collections.amazon.aws.module_utils.urls is believed to be unused and has been deprecated and will be removed in release 7.0.0.
- s3_bucket - The S3_URL alias for the s3_url option has been deprecated and will be removed in release 5.0.0 (https://github.com/ansible-collections/community.aws/pull/795).
- s3_object - Support for creation and deletion of S3 buckets has been deprecated. Please use the amazon.aws.s3_bucket module to create and delete buckets (https://github.com/ansible-collections/amazon.aws/pull/869).

cisco.ios

- Deprecated ios_linkagg_module in favor of ios_lag_interfaces.

cisco.mso

- The mso_schema_template_contract_filter contract_filter_type attribute is deprecated. The value is now deduced from filter_type.

community.aws

- aws_acm - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- aws_codebuild - The `tags` parameter currently uses a non-standard format and has been deprecated. In release 6.0.0 this parameter will accept a simple key/value pair dictionary instead of the current list of dictionaries. It is recommended to migrate to using the resource_tags parameter which already accepts the simple dictionary format (https://github.com/ansible-collections/community.aws/pull/1221).
- aws_glue_connection - the `connection_parameters` return key has been deprecated and will be removed in a release after 2024-06-01, it is being replaced by the `raw_connection_parameters` key (https://github.com/ansible-collections/community.aws/pull/518).
- aws_kms - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- cloudfront_distribution - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- community.aws collection - due to the AWS SDKs announcing the end of support for Python less than 3.7 (https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/) support for Python less than 3.7 by this collection has been deprecated and will be removed in a release after 2023-05-31 (https://github.com/ansible-collections/community.aws/pull/1361).
- ec2_vpc_vpn - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- iam_policy - the `policies` return value has been renamed `policy_names` and will be removed in a release after 2024-08-01, both values are currently returned (https://github.com/ansible-collections/community.aws/pull/1375).
- lambda_info - The `function` return key returns a dictionary of dictionaries and has been deprecated. In a release after 2025-01-01, this key will be removed in favor of `functions`, which returns a list of dictionaries (https://github.com/ansible-collections/community.aws/pull/1239).
- rds_param_group - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- route53_health_check - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- route53_info - The CamelCase return values for `DelegationSets`, `CheckerIpRanges`, and `HealthCheck` have been deprecated, in the future release you must use snake_case return values `delegation_sets`, `checker_ip_ranges`, and `health_check` instead respectively" (https://github.com/ansible-collections/community.aws/pull/1322).

- route53_info - The CamelCase return values for `HostedZones`, `ResourceRecordSets`, and `HealthChecks` have been deprecated, in the future release you must use snake_case return values `hosted_zones`, `resource_record_sets`, and `health_checks` instead respectively".
- route53_zone - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.
- sqs_queue - the current default value of `False` for `purge_tags` has been deprecated and will be updated in release 5.0.0 to `True`.

community.crypto

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.crypto 3.0.0). Some modules might still work with these versions afterwards, but we will no longer keep compatibility code that was needed to support them (https://github.com/ansible-collections/community.crypto/pull/460).

community.docker

- Support for Docker API version 1.20 to 1.24 has been deprecated and will be removed in community.docker 3.0.0. The first Docker version supporting API version 1.25 was Docker 1.13, released in January 2017. This affects the modules `docker_container`, `docker_container_exec`, `docker_container_info`, `docker_compose`, `docker_login`, `docker_image`, `docker_image_info`, `docker_image_load`, `docker_host_info`, `docker_network`, `docker_network_info`, `docker_node_info`, `docker_swarm_info`, `docker_swarm_service`, `docker_swarm_service_info`, `docker_volume_info`, and `docker_volume`, whose minimally supported API version is between 1.20 and 1.24 (https://github.com/ansible-collections/community.docker/pull/396).
- Support for Python 2.6 is deprecated and will be removed in the next major release (community.docker 3.0.0). Some modules might still work with Python 2.6, but we will no longer try to ensure compatibility (https://github.com/ansible-collections/community.docker/pull/388).
- docker_container - the `ignore_image` option is deprecated and will be removed in community.docker 4.0.0. Use `image: ignore` in `comparisons` instead (https://github.com/ansible-collections/community.docker/pull/487).
- docker_container - the `purge_networks` option is deprecated and will be removed in community.docker 4.0.0. Use `networks: strict` in `comparisons` instead, and make sure to provide `networks`, with value `[]` if all networks should be removed (https://github.com/ansible-collections/community.docker/pull/487).

community.general

- ArgFormat module utils - deprecated along `CmdMixin`, in favor of the `cmd_runner_fmt` module util (https://github.com/ansible-collections/community.general/pull/5370).
- CmdMixin module utils - deprecated in favor of the `CmdRunner` module util (https://github.com/ansible-collections/community.general/pull/5370).
- CmdModuleHelper module utils - deprecated in favor of the `CmdRunner` module util (https://github.com/ansible-collections/community.general/pull/5370).
- CmdStateModuleHelper module utils - deprecated in favor of the `CmdRunner` module util (https://github.com/ansible-collections/community.general/pull/5370).
- cmd_runner module utils - deprecated `fmt` in favour of `cmd_runner_fmt` as the parameter format object (https://github.com/ansible-collections/community.general/pull/4777).

- django_manage - support for Django releases older than 4.1 has been deprecated and will be removed in community.general 9.0.0 (https://github.com/ansible-collections/community.general/pull/5400).
- django_manage - support for the commands `cleanup`, `syncdb` and `validate` that have been deprecated in Django long time ago will be removed in community.general 9.0.0 (https://github.com/ansible-collections/community.general/pull/5400).
- django_manage - the behavior of "creating the virtual environment when missing" is being deprecated and will be removed in community.general version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5405).
- gconftool2 - deprecates `state=get` in favor of using the module `gconftool2_info` (https://github.com/ansible-collections/community.general/pull/4778).
- lxc_container - the module will no longer make any effort to support Python 2 (https://github.com/ansible-collections/community.general/pull/5304).
- newrelic_deployment - `appname` and `environment` are no longer valid options in the v2 API. They will be removed in community.general 7.0.0 (https://github.com/ansible-collections/community.general/pull/5341).
- proxmox - deprecated the current `unprivileged` default value, will be changed to `true` in community.general 7.0.0 (https://github.com/pull/5224).
- xfconf - deprecated parameter `disable_facts`, as since version 4.0.0 it only allows value `true` (https://github.com/ansible-collections/community.general/pull/4520).

community.hashi_vault

- vault_kv2_get lookup - the `engine_mount_point option` in the `vault_kv2_get` lookup only will change its default from `kv` to `secret` in community.hashi_vault version 4.0.0 (https://github.com/ansible-collections/community.hashi_vault/issues/279).

1.3.3 Ansible 6 Porting Guide

Ansible 6 is based on Ansible-core 2.13.

We suggest you read this page along with the Ansible 6 Changelog to understand what updates you may need to make.

Playbook

- Templating - You can no longer perform arithmetic and concatenation operations outside of the jinja template. The following statement will need to be rewritten to produce `[1, 2]`:

```
- name: Prior to 2.13
  debug:
    msg: '[1] + {{ [2] }}'

- name: 2.13 and forward
  debug:
    msg: '{{ [1] + [2] }}'
```

- The return value of the `__repr__` method of an undefined variable represented by the `AnsibleUndefined` object changed. `{{ '%r'|format(undefined_variable) }}` returns `AnsibleUndefined(hint=None, obj=missing, name='undefined_variable')` in 2.13 as opposed to just `AnsibleUndefined` in versions 2.12 and prior.
- The `finalize` method is no longer exposed in the globals for use in templating. To convert `None` to an empty string the following expression can be used: `{{ value if value is not none }}`.

Command Line

No notable changes

Deprecated

No notable changes

Modules

- To use ansible-core 2.13 for module execution, you must use Python 2 version 2.7 or Python 3 version 3.5 or newer. Any code utilizing `ansible.module_utils.basic` will not function with lower Python versions.

Modules removed

The following modules no longer exist:

- No notable changes

Deprecation notices

No notable changes

Noteworthy module changes

No notable changes

Breaking Changes

- `ansible.module_utils.urls.fetch_url` will now return the captured `HTTPError` exception as `r.HTTPError` is a response like object that can offer more information to module authors. Modules should rely on `info['status'] >= 400` to determine if there was a failure, instead of using `r is None` or catching `AttributeError` when attempting `r.read()`.

Plugins

No notable changes

Porting custom scripts

No notable changes

Networking

No notable changes

Porting Guide for v6.7.0

Known Issues

community.routeros

- api_modify - when limits for entries in `queue tree` are defined as human readable - for example `25M` -, the configuration will be correctly set in ROS, but the module will indicate the item is changed on every run even when there was no change done. This is caused by the ROS API which returns the number in bytes - for example `25000000` (which is inconsistent with the CLI behavior). In order to mitigate that, the limits have to be defined in bytes (those will still appear as human readable in the ROS CLI) (https://github.com/ansible-collections/community.routeros/pull/131).
- api_modify, api_info - `routing ospf area`, `routing ospf area range`, `routing ospf instance`, `routing ospf interface-template` paths are not fully implemeted for ROS6 due to the significat changes between ROS6 and ROS7 (https://github.com/ansible-collections/community.routeros/pull/131).

Major Changes

cisco.meraki

- meraki_mr_l7_firewall - New module
- meraki_webhook_payload_template - New module

community.zabbix

- all modules are opting away from zabbix-api and using httpapi ansible.netcommon plugin. We will support zabbix-api for backwards compatibility until next major release. See our README.md for more information about how to migrate
- zabbix_agent and zabbix_proxy roles are opting away from zabbix-api and use httpapi ansible.netcommon plugin. We will support zabbix-api for backwards compatibility until next major release. See our README.md for more information about how to migrate

containers.podman

- New become plugin - podman_unshare
- Podman generate systemd module

fortinet.fortimanager

- Fix compatibility issue for ansible 2.9.x and ansible-base 2.10.x.
- support Ansible changelogs.

fortinet.fortios

- Support FortiOS v7.0.6, v7.0.7, v7.0.8, v7.2.1, v7.2.2.

Deprecated Features

community.general

- Please note that some tools, like the VScode plugin (https://github.com/ansible/vscode-ansible/issues/573), or `ansible-doc --list --type module`, suggest to replace the correct FQCNs for modules and actions in community.general with internal names that have more than three components. For example, `community.general.ufw` is suggested to be replaced by `community.general.system.ufw`. While these longer names do work, they are considered **internal names** by the collection and are subject to change and be removed at all time. They **will** be removed in community.general 6.0.0 and result in deprecation messages. Avoid using these internal names, and use general three-component FQCNs (`community.general.<name_of_module>`) instead (https://github.com/ansible-collections/community.general/pull/5373).

Porting Guide for v6.6.0

Added Collections

- lowlydba.sqlserver (version 1.0.4)

Known Issues

community.routeros

- The `community.routeros.command` module claims to support check mode. Since it cannot judge whether the commands executed modify state or not, this behavior is incorrect. Since this potentially breaks existing playbooks, we will not change this behavior until community.routeros 3.0.0.

Breaking Changes

community.general

- newrelic_deployment - `revision` is required for v2 API (https://github.com/ansible-collections/community.general/pull/5341).

Major Changes

community.general

- newrelic_deployment - removed New Relic v1 API, added support for v2 API (https://github.com/ansible-collections/community.general/pull/5341).

fortinet.fortimanager

- Many fixes for Ansible sanity test warnings & errors.
- Support FortiManager Schema 7.2.0 , 98 new modules

Deprecated Features

- The mellanox.onyx collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/136).

cisco.mso

- The mso_schema_template_contract_filter contract_filter_type attribute is deprecated. The value is now deduced from filter_type.

community.general

- ArgFormat module utils - deprecated along `CmdMixin`, in favor of the `cmd_runner_fmt` module util (https://github.com/ansible-collections/community.general/pull/5370).
- CmdMixin module utils - deprecated in favor of the `CmdRunner` module util (https://github.com/ansible-collections/community.general/pull/5370).
- CmdModuleHelper module utils - deprecated in favor of the `CmdRunner` module util (https://github.com/ansible-collections/community.general/pull/5370).
- CmdStateModuleHelper module utils - deprecated in favor of the `CmdRunner` module util (https://github.com/ansible-collections/community.general/pull/5370).
- django_manage - support for Django releases older than 4.1 has been deprecated and will be removed in community.general 9.0.0 (https://github.com/ansible-collections/community.general/pull/5400).
- django_manage - support for the commands `cleanup`, `syncdb` and `validate` that have been deprecated in Django long time ago will be removed in community.general 9.0.0 (https://github.com/ansible-collections/community.general/pull/5400).
- django_manage - the behavior of "creating the virtual environment when missing" is being deprecated and will be removed in community.general version 9.0.0 (https://github.com/ansible-collections/community.general/pull/5405).
- newrelic_deployment - `appname` and `environment` are no longer valid options in the v2 API. They will be removed in community.general 7.0.0 (https://github.com/ansible-collections/community.general/pull/5341).

Porting Guide for v6.5.0

Major Changes

infoblox.nios_modules

- Feature for extra layer security , with *cert* and *key* parameters in playbooks for authenticating using certificate and key `*.pem` file absolute path #154
- Fix to remove issue causing due to template attr in deleting network using Ansible module nios network #147

Deprecated Features

- The dellemc.os10 collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/134).
- The dellemc.os6 collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/132).

- The dellemc.os9 collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/133).

community.general

- lxc_container - the module will no longer make any effort to support Python 2 (https://github.com/ansible-collections/community.general/pull/5304).

Porting Guide for v6.4.0

Added Collections

- inspur.ispim (version 1.0.1)
- vultr.cloud (version 1.1.0)

Deprecated Features

community.general

- proxmox - deprecated the current `unprivileged` default value, will be changed to `true` in community.general 7.0.0 (https://github.com/pull/5224).

Porting Guide for v6.3.0

Major Changes

community.mysql

- mysql_db - the `pipefail` argument's default value will be changed to `true` in community.mysql 4.0.0. If your target machines do not use `bash` as a default interpreter, set `pipefail` to `false` explicitly. However, we strongly recommend setting up `bash` as a default and `pipefail=true` as it will protect you from getting broken dumps you don't know about (https://github.com/ansible-collections/community.mysql/issues/407).

fortinet.fortios

- Support Diff feature in check_mode.
- Support Fortios 7.2.0.

Deprecated Features

- The google.cloud collection is considered unmaintained and will be removed from Ansible 8 if no one starts maintaining it again before Ansible 8. See the removal process for details on how this works (https://github.com/ansible-community/community-topics/issues/105).
- The servicenow.servicenow collection has been deprecated by its maintainers (https://github.com/ServiceNowITOM/servicenow-ansible/pull/69) and will be removed from Ansible 7. It can still be installed manually, but it is suggested to switch to servicenow.itsm instead (https://github.com/ansible-community/community-topics/issues/124).

Porting Guide for v6.2.0

Added Collections

- ibm.spectrum_virtualize (version 1.9.0)

Known Issues

netapp.ontap

- na_ontap_snapshot - added documentation to use UTC format for `expiry_time`.

Major Changes

community.postgresql

- postgresql_user - the `groups` argument has been deprecated and will be removed in `community.postgresql 3.0.0`. Please use the `postgresql_membership` module to specify group/role memberships instead (https://github.com/ansible-collections/community.postgresql/issues/277).

Deprecated Features

community.hashi_vault

- vault_kv2_get lookup - the `engine_mount_point option` in the `vault_kv2_get` lookup only will change its default from `kv` to `secret` in community.hashi_vault version 4.0.0 (https://github.com/ansible-collections/community.hashi_vault/issues/279).

Porting Guide for v6.1.0

Added Collections

- purestorage.fusion (version 1.0.2)

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) The module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_application_alerts_smtp - Issue(212310) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_application_alerts_syslog - Issue(215374) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_device_local_access_configuration - Issue(215035) - The module reports `Successfully updated the local access setting` if an unsupported value is provided for the parameter timeout_limit. However, this value is not actually applied on OpenManage Enterprise Modular.
- ome_device_local_access_configuration - Issue(217865) - The module does not display a proper error message if an unsupported value is provided for the user_defined and lcd_language parameters.
- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_power_settings - Issue(212679) - The module displays the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_device_quick_deploy - Issue(216352) - The module does not display a proper error message if an unsupported value is provided for the ipv6_prefix_length and vlan_id parameters.
- ome_smart_fabric_uplink - Issue(186024) - The module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Major Changes

chocolatey.chocolatey

- win_chocolatey - Added bootstrap_script option to allow users to target a script URL for installing Chocolatey on clients.
- win_chocolatey_facts - Added outdated packages list to data returned.

infoblox.nios_modules

- Update *text* field of TXT Record #128
- Update operation using *old_name* and *new_name* for the object with dummy name in *old_name* (which does not exist in system) will not create a new object in the system. An error will be thrown stating the object does not exist in the system #129

Deprecated Features

cisco.ios

- Deprecated ios_linkagg_module in favor of ios_lag_interfaces.

community.aws

- aws_codebuild - The `tags` parameter currently uses a non-standard format and has been deprecated. In release 6.0.0 this parameter will accept a simple key/value pair dictionary instead of the current list of dictionaries. It is recommended to migrate to using the resource_tags parameter which already accepts the simple dictionary format (https://github.com/ansible-collections/community.aws/pull/1221).
- route53_info - The CamelCase return values for `HostedZones`, `ResourceRecordSets`, and `HealthChecks` have been deprecated, in the future release you must use snake_case return values `hosted_zones`, `resource_record_sets`, and `health_checks` instead respectively".

community.crypto

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.crypto 3.0.0). Some modules might still work with these versions afterwards, but we will no longer keep compatibility code that was needed to support them (https://github.com/ansible-collections/community.crypto/pull/460).

community.docker

- Support for Docker API version 1.20 to 1.24 has been deprecated and will be removed in community.docker 3.0.0. The first Docker version supporting API version 1.25 was Docker 1.13, released in January 2017. This affects the modules `docker_container`, `docker_container_exec`, `docker_container_info`, `docker_compose`, `docker_login`, `docker_image`, `docker_image_info`, `docker_image_load`, `docker_host_info`, `docker_network`, `docker_network_info`, `docker_node_info`, `docker_swarm_info`, `docker_swarm_service`, `docker_swarm_service_info`, `docker_volume_info`, and `docker_volume`, whose minimally supported API version is between 1.20 and 1.24 (https://github.com/ansible-collections/community.docker/pull/396).
- Support for Python 2.6 is deprecated and will be removed in the next major release (community.docker 3.0.0). Some modules might still work with Python 2.6, but we will no longer try to ensure compatibility (https://github.com/ansible-collections/community.docker/pull/388).

community.general

- cmd_runner module utils - deprecated `fmt` in favour of `cmd_runner_fmt` as the parameter format object (https://github.com/ansible-collections/community.general/pull/4777).

Porting Guide for v6.0.0

Added Collections

- cisco.dnac (version 6.4.0)
- community.sap (version 1.0.0)
- community.sap_libs (version 1.1.0)
- vmware.vmware_rest (version 2.1.5)

Known Issues

Ansible-core

- get_url - document `check_mode` correctly with unreliable changed status (https://github.com/ansible/ansible/issues/65687).

ansible.netcommon

- eos - When using eos modules on Ansible 2.9, tasks will occasionally fail with `import_modules` enabled. This can be avoided by setting `import_modules: no`

community.general

- pacman - `update_cache` cannot differentiate between up to date and outdated package lists and will report `changed` in both situations (https://github.com/ansible-collections/community.general/pull/4318).
- pacman - binaries specified in the `executable` parameter must support `--print-format` in order to be used by this module. In particular, AUR helper `yay` is known not to currently support it (https://github.com/ansible-collections/community.general/pull/4312).

dellemc.openmanage

- idrac_user - Issue(192043) The module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_application_alerts_smtp - Issue(212310) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_application_alerts_syslog - Issue(215374) - The module does not provide a proper error message if the destination_address is more than 255 characters.

- ome_application_console_preferences - Issue(224690) - The module does not display a proper error message when an unsupported value is provided for the parameters report_row_limit, email_sender_settings, and metric_collection_settings, and the value is applied on OpenManage Enterprise.
- ome_device_local_access_configuration - Issue(215035) - The module reports `Successfully updated the local access setting` if an unsupported value is provided for the parameter timeout_limit. However, this value is not actually applied on OpenManage Enterprise Modular.
- ome_device_local_access_configuration - Issue(217865) - The module does not display a proper error message if an unsupported value is provided for the user_defined and lcd_language parameters.
- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_power_settings - Issue(212679) - The module displays the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_device_power_settings - Issue(212679) - The module errors out with the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_device_power_settings - Issue(212679) - The module errors out with the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_device_quick_deploy - Issue(216352) - The module does not display a proper error message if an unsupported value is provided for the ipv6_prefix_length and vlan_id parameters.
- ome_smart_fabric_uplink - Issue(186024) - The module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

purestorage.flasharray

- purefa_admin - Once *max_login* and *lockout* have been set there is currently no way to rest these to zero except through the FlashArray GUI

Breaking Changes

Ansible-core

- Module Python Dependency - Drop support for Python 2.6 in module execution.
- Templating - it is no longer allowed to perform arithmetic and concatenation operations outside of the jinja template (https://github.com/ansible/ansible/pull/75587)
- The `finalize` method is no longer exposed in the globals for use in templating.

amazon.aws

- aws_caller_facts - Remove deprecated `aws_caller_facts` alias. Please use `aws_caller_info` instead.
- cloudformation_facts - Remove deprecated `cloudformation_facts` alias. Please use `cloudformation_info` instead.
- ec2_ami_facts - Remove deprecated `ec2_ami_facts` alias. Please use `ec2_ami_info` instead.
- ec2_eni_facts - Remove deprecated `ec2_eni_facts` alias. Please use `ec2_eni_info` instead.
- ec2_group_facts - Remove deprecated `ec2_group_facts` alias. Please use `ec2_group_info` instead.
- ec2_instance_facts - Remove deprecated `ec2_instance_facts` alias. Please use `ec2_instance_info` instead.
- ec2_snapshot_facts - Remove deprecated `ec2_snapshot_facts` alias. Please use `ec2_snapshot_info` instead.
- ec2_vol_facts - Remove deprecated `ec2_vol_facts` alias. Please use `ec2_vol_info` instead.
- ec2_vpc_dhcp_option_facts - Remove deprecated `ec2_vpc_dhcp_option_facts` alias. Please use `ec2_vpc_dhcp_option_info` instead.
- ec2_vpc_endpoint_facts - Remove deprecated `ec2_vpc_endpoint_facts` alias. Please use `ec2_vpc_endpoint_info` instead.
- ec2_vpc_igw_facts - Remove deprecated `ec2_vpc_igw_facts` alias. Please use `ec2_vpc_igw_info` instead.
- ec2_vpc_nat_gateway_facts - Remove deprecated `ec2_vpc_nat_gateway_facts` alias. Please use `ec2_vpc_nat_gateway_info` instead.
- ec2_vpc_net_facts - Remove deprecated `ec2_vpc_net_facts` alias. Please use `ec2_vpc_net_info` instead.
- ec2_vpc_route_table_facts - Remove deprecated `ec2_vpc_route_table_facts` alias. Please use `ec2_vpc_route_table_info` instead.
- ec2_vpc_subnet_facts - Remove deprecated `ec2_vpc_subnet_facts` alias. Please use `ec2_vpc_subnet_info` instead.

ansible.netcommon

- httpapi - Change default value of `import_modules` option from `no` to `yes`
- netconf - Change default value of `import_modules` option from `no` to `yes`
- network_cli - Change default value of `import_modules` option from `no` to `yes`

arista.eos

- eos_command - new suboption `version` of parameter `command`, which controls the JSON response version. Previously the value was assumed to be "latest" for network_cli and "1" for httpapi, but the default will now be "latest" for both connections. This option is also available for use in modules making their own device requests with `plugins.module_utils.network.eos.eos.run_commands()` with the same new default behavior. (https://github.com/ansible-collections/arista.eos/pull/258).
- httpapi - the `eos_use_sessions` option is now a boolean instead of an integer.

community.aws

- aws_acm_facts - Remove deprecated alias `aws_acm_facts`. Please use `aws_acm_info` instead.
- aws_kms_facts - Remove deprecated alias `aws_kms_facts`. Please use `aws_kms_info` instead.
- aws_kms_info - Deprecated `keys_attr` field is now ignored (https://github.com/ansible-collections/community.aws/pull/838).
- aws_region_facts - Remove deprecated alias `aws_region_facts`. Please use `aws_region_info` instead.
- aws_s3_bucket_facts - Remove deprecated alias `aws_s3_bucket_facts`. Please use `aws_s3_bucket_info` instead.
- aws_sgw_facts - Remove deprecated alias `aws_sgw_facts`. Please use `aws_sgw_info` instead.
- aws_waf_facts - Remove deprecated alias `aws_waf_facts`. Please use `aws_waf_info` instead.
- cloudfront_facts - Remove deprecated alias `cloudfront_facts`. Please use `cloudfront_info` instead.
- cloudwatchlogs_log_group_facts - Remove deprecated alias `cloudwatchlogs_log_group_facts`. Please use `cloudwatchlogs_log_group_info` instead.
- dynamodb_table - deprecated updates currently ignored for primary keys and global_all indexes will now result in a failure. (https://github.com/ansible-collections/community.aws/pull/837).
- ec2_asg_facts - Remove deprecated alias `ec2_asg_facts`. Please use `ec2_asg_info` instead.
- ec2_customer_gateway_facts - Remove deprecated alias `ec2_customer_gateway_facts`. Please use `ec2_customer_gateway_info` instead.
- ec2_eip_facts - Remove deprecated alias `ec2_eip_facts`. Please use `ec2_eip_info` instead.
- ec2_elb_facts - Remove deprecated alias `ec2_elb_facts`. Please use `ec2_elb_info` instead.
- ec2_elb_info - The `ec2_elb_info` module has been removed. Please use `` the ``elb_classic_lb_info `` module.
- ec2_lc_facts - Remove deprecated alias `ec2_lc_facts`. Please use `ec2_lc_info` instead.
- ec2_placement_group_facts - Remove deprecated alias `ec2_placement_group_facts`. Please use `ec2_placement_group_info` instead.
- ec2_vpc_nacl_facts - Remove deprecated alias `ec2_vpc_nacl_facts`. Please use `ec2_vpc_nacl_info` instead.
- ec2_vpc_peering_facts - Remove deprecated alias `ec2_vpc_peering_facts`. Please use `ec2_vpc_peering_info` instead.
- ec2_vpc_route_table_facts - Remove deprecated alias `ec2_vpc_route_table_facts`. Please use `ec2_vpc_route_table_info` instead.
- ec2_vpc_vgw_facts - Remove deprecated alias `ec2_vpc_vgw_facts`. Please use `ec2_vpc_vgw_info` instead.
- ec2_vpc_vpn_facts - Remove deprecated alias `ec2_vpc_vpn_facts`. Please use `ec2_vpc_vpn_info` instead.
- ecs_service_facts - Remove deprecated alias `ecs_service_facts`. Please use `ecs_service_info` instead.
- ecs_taskdefinition_facts - Remove deprecated alias `ecs_taskdefinition_facts`. Please use `ecs_taskdefinition_info` instead.
- efs_facts - Remove deprecated alias `efs_facts`. Please use `efs_info` instead.
- elasticache_facts - Remove deprecated alias `elasticache_facts`. Please use `elasticache_info` instead.
- elb_application_lb_facts - Remove deprecated alias `elb_application_lb_facts`. Please use `elb_application_lb_info` instead.

- elb_classic_lb_facts - Remove deprecated alias `elb_classic_lb_facts`. Please use `elb_classic_lb_info` instead.
- elb_target_facts - Remove deprecated alias `elb_target_facts`. Please use `elb_target_info` instead.
- elb_target_group_facts - Remove deprecated alias `elb_target_group_facts`. Please use `elb_target_group_info` instead.
- iam - Removed deprecated `community.aws.iam` module. Please use `community.aws.iam_user`, `community.aws.iam_access_key` or `community.aws.iam_group` (https://github.com/ansible-collections/community.aws/pull/839).
- iam_cert_facts - Remove deprecated alias `iam_cert_facts`. Please use `iam_cert_info` instead.
- iam_mfa_device_facts - Remove deprecated alias `iam_mfa_device_facts`. Please use `iam_mfa_device_info` instead.
- iam_role_facts - Remove deprecated alias `iam_role_facts`. Please use `iam_role_info` instead.
- iam_server_certificate_facts - Remove deprecated alias `iam_server_certificate_facts`. Please use `iam_server_certificate_info` instead.
- lambda_facts - Remove deprecated module lambda_facts``. Please use `lambda_info` instead.
- rds - Removed deprecated `community.aws.rds` module. Please use `community.aws.rds_instance` (https://github.com/ansible-collections/community.aws/pull/839).
- rds_instance_facts - Remove deprecated alias `rds_instance_facts`. Please use `rds_instance_info` instead.
- rds_snapshot_facts - Remove deprecated alias `rds_snapshot_facts`. Please use `rds_snapshot_info` instead.
- redshift_facts - Remove deprecated alias `redshift_facts`. Please use `redshift_info` instead.
- route53_facts - Remove deprecated alias `route53_facts`. Please use `route53_info` instead.

community.general

- Parts of this collection do not work with ansible-core 2.11 on Python 3.12+. Please either upgrade to ansible-core 2.12+, or use Python 3.11 or earlier (https://github.com/ansible-collections/community.general/pull/3988).
- The symbolic links used to implement flatmapping for all modules were removed and replaced by `meta/runtime.yml` redirects. This effectively breaks compatibility with Ansible 2.9 for all modules (without using their "long" names, which is discouraged and which can change without previous notice since they are considered an implementation detail) (https://github.com/ansible-collections/community.general/pull/4548).
- a_module test plugin - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- archive - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- git_config - remove Ansible 2.9 and early ansible-base 2.10 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- java_keystore - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- lists_mergeby and groupby_as_dict filter plugins - adjust filter plugin filename. This change is not visible to end-users, it only affects possible other collections importing Python paths (https://github.com/ansible-collections/community.general/pull/4625).

- lists_mergeby filter plugin - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- maven_artifact - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- memcached cache plugin - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- path_join filter plugin shim - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- redis cache plugin - remove Ansible 2.9 compatibility code (https://github.com/ansible-collections/community.general/pull/4548).
- yarn - remove unsupported and unnecessary `--no-emoji` flag (https://github.com/ansible-collections/community.general/pull/4662).

community.mysql

- mysql_replication - remove `Is_Slave` and `Is_Master` return values (were replaced with `Is_Primary` and `Is_Replica` (https://github.com/ansible-collections /community.mysql/issues/145).
- mysql_replication - remove the mode options values containing `master`/`slave` and the master_use_gtid option `slave_pos` (were replaced with corresponding `primary`/`replica` values) (https://github.com/ansible-collections/community.mysql/issues/145).
- mysql_user - remove support for the *REQUIRESSL* special privilege as it has ben superseded by the *tls_requires* option (https://github.com/ansible-collections/community.mysql/discussions/121).
- mysql_user - validate privileges using database engine directly (https://github.com/ansible-collections/community.mysql/issues/234 https://github.com/ansible-collections/community.mysql/pull/243). Do not validate privileges in this module anymore.

community.vmware

- The collection now requires at least ansible-core 2.11.0. Ansible 3 and before, and ansible-base versions are no longer supported.
- vmware_cluster_drs - The default for `enable` has been changed from `false` to `true`.
- vmware_cluster_drs - The parameter alias `enable_drs` has been removed, use `enable` instead.
- vmware_cluster_ha - The default for `enable` has been changed from `false` to `true`.
- vmware_cluster_ha - The parameter alias `enable_ha` has been removed, use `enable` instead.
- vmware_cluster_vsan - The default for `enable` has been changed from `false` to `true`.
- vmware_cluster_vsan - The parameter alias `enable_vsan` has been removed, use `enable` instead.
- vmware_guest - Virtualization Based Security has some requirements (`nested_virt`, `secure_boot` and `iommu`) that the module silently enabled. They have to be enabled explicitly now.

dellemc.openmanage

- HTTPS SSL certificate validation is a **breaking change** and will require modification in the existing playbooks. Please refer to SSL Certificate Validation section in the README.md for modification to existing playbooks.

theforeman.foreman

- Set use_reports_api default value to true for the inventory plugin
- Support for Ansible 2.8 is removed

Major Changes

- Add a `ansible-community` CLI tool that allows to print the version of the Ansible community distribution. Use `ansible-community --version` to print this version.

Ansible-core

- Jinja2 Controller Requirement - Jinja2 3.0.0 or newer is required for the control node (the machine that runs Ansible) (https://github.com/ansible/ansible/pull/75881)
- Templating - remove `safe_eval` in favor of using `NativeEnvironment` but utilizing `literal_eval` only in cases when `safe_eval` was used (https://github.com/ansible/ansible/pull/75587)

amazon.aws

- amazon.aws collection - The amazon.aws collection has dropped support for `botocore<1.19.0` and `boto3<1.16.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/amazon.aws/pull/574).

ansible.netcommon

- cli_parse - this module has been moved to the ansible.utils collection. `ansible.netcommon.cli_parse` will continue to work to reference the module in its new location, but this redirect will be removed in a future release
- network_cli - Change default value of *ssh_type* option from *paramiko* to *auto*. This value will use libssh if the ansible-pylibssh module is installed, otherwise will fallback to paramiko.

arista.eos

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.
- *eos_facts* - change default gather_subset to *min* from *!config* (https://github.com/ansible-collections/arista.eos/issues/306).

chocolatey.chocolatey

- win_chocolatey - Added choco_args option to pass additional arguments directly to Chocolatey.

cisco.asa

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.

cisco.ios

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.
- facts - default value for gather_subset is changed to min instead of !config.

cisco.iosxr

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.
- *facts* - default value for *gather_subset* is changed to min instead of !config.

cisco.ise

- Update ciscoisesdk requirement to 1.2.0
- anc_endpoint_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- anc_policy_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- backup_last_status_info - change return value, it returns response content.
- device_administration_authentication_rules - deletes parameter identitySourceId.
- device_administration_authentication_rules_info - change return value, it returns response content.
- device_administration_authorization_rules_info - change return value, it returns response content.
- device_administration_conditions - deletes parameter attributeId.
- device_administration_conditions_for_authentication_rule_info - change return value, it returns response content.
- device_administration_conditions_for_authorization_rule_info - change return value, it returns response content.
- device_administration_conditions_for_policy_set_info - change return value, it returns response content.
- device_administration_conditions_info - change return value, it returns response content.
- device_administration_dictionary_attributes_authentication_info - change return value, it returns response content.
- device_administration_dictionary_attributes_authorization_info - change return value, it returns response content.

- device_administration_dictionary_attributes_policy_set_info - change return value, it returns response content.
- device_administration_global_exception_rules_info - change return value, it returns response content.
- device_administration_network_conditions_info - change return value, it returns response content.
- device_administration_time_date_conditions - deletes parameter attributeId.
- device_administration_time_date_conditions_info - change return value, it returns response content.
- egress_matrix_cell_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- network_access_authentication_rules - deletes parameter identitySourceId.
- network_access_conditions - deletes parameter attributeId.
- network_access_time_date_conditions - deletes parameter attributeId.
- node_deployment - update parameters.
- node_deployment_info - add filter and filterType parameters.
- node_group - fixes response recollection.
- node_group_info - fixes response recollection.
- repository_files_info - change return value, it returns response content.
- repository_info - change return value, it returns response content.
- sg_acl_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sg_mapping_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sg_mapping_group_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sg_mapping_group_info - change return value, it returns BulkStatus content.
- sg_to_vn_to_vlan_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sgt - change generationId type from int to str.
- sgt_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sxp_connections_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sxp_local_bindings_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- sxp_vpns_bulk_monitor_status_info - change return value, it returns BulkStatus content.
- system_certificate - new parameters portalTagTransferForSameSubject and roleTransferForSameSubject.
- system_certificate - portalTagTransferForSameSubject parameter renamed to allowPortalTagTransferForSameSubject.
- system_certificate - roleTransferForSameSubject parameter renamed to allowRoleTransferForSameSubject.
- system_certificate_import - new parameters portalTagTransferForSameSubject and roleTransferForSameSubject.
- system_certificate_import - portalTagTransferForSameSubject parameter renamed to allowPortalTagTransferForSameSubject.
- system_certificate_import - roleTransferForSameSubject parameter renamed to allowRoleTransferForSameSubject.
- trustsec_nbar_app_info - change type from str to list.
- trustsec_vn_info - change type from str to list.

cisco.meraki

- meraki_mr_radio - New module

cisco.nxos

- The minimum required ansible.netcommon version has been bumped to v2.6.1.
- Updated base plugin references to ansible.netcommon.
- *nxos_facts* - change default gather_subset to *min* from *!config* (https://github.com/ansible-collections/cisco.nxos/issues/418).
- nxos_file_copy has been rewritten as a module. This change also removes the dependency on pexpect for file_pull operation. Since this now uses AnsibleModule class for argspec validation, the validation messages will be slightly different. Expect changes in the return payload in some cases. All functionality remains unchanged.

community.aws

- community.aws collection - The community.aws collection has dropped support for `botocore<1.19.0` and `boto3<1.16.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/community.aws/pull/809).
- s3_bucket_notifications - refactor module to support SNS / SQS targets as well as the existing support for Lambda functions (https://github.com/ansible-collections/community.aws/issues/140).

community.general

- The community.general collection no longer supports Ansible 2.9 and ansible-base 2.10. While we take no active measures to prevent usage, we will remove a lot of compatibility code and other compatility measures that will effectively prevent using most content from this collection with Ansible 2.9, and some content of this collection with ansible-base 2.10. Both Ansible 2.9 and ansible-base 2.10 will very soon be End of Life and if you are still using them, you should consider upgrading to ansible-core 2.11 or later as soon as possible (https://github.com/ansible-collections/community.general/pull/4548).

community.mysql

- The community.mysql collection no longer supports `Ansible 2.9` and `ansible-base 2.10`. While we take no active measures to prevent usage and there are no plans to introduce incompatible code to the modules, we will stop testing against `Ansible 2.9` and `ansible-base 2.10`. Both will very soon be End of Life and if you are still using them, you should consider upgrading to the `latest Ansible / ansible-core 2.11 or later` as soon as possible (https://github.com/ansible-collections/community.mysql/pull/343).

community.network

- The community.network collection no longer supports Ansible 2.9 and ansible-base 2.10. While we take no active measures to prevent usage, we will remove compatibility code and other compatility measures that will effectively prevent using most content from this collection with Ansible 2.9, and some content of this collection with ansible-base 2.10. Both Ansible 2.9 and ansible-base 2.10 will very soon be End of Life and if you are still using them, you should consider upgrading to ansible-core 2.11 or later as soon as possible (https://github.com/ansible-collections/community.network/pull/426).

community.postgresql

- The community.postgresql collection no longer supports `Ansible 2.9` and `ansible-base 2.10`. While we take no active measures to prevent usage and there are no plans to introduce incompatible code to the modules, we will stop testing against `Ansible 2.9` and `ansible-base 2.10`. Both will very soon be End of Life and if you are still using them, you should consider upgrading to the `latest Ansible / ansible-core 2.11 or later` as soon as possible (https://github.com/ansible-collections/community.postgresql/pull/245).
- postgresql_privs - the `usage_on_types` feature have been deprecated and will be removed in `community.postgresql 3.0.0`. Please use the `type` option with the `type` value to explicitly grant/revoke privileges on types (https://github.com/ansible-collections/community.postgresql/issues/207).
- postgresql_query - the `path_to_script` and `as_single_query` options as well as the `query_list` and `query_all_results` return values have been deprecated and will be removed in `community.postgresql 3.0.0`. Please use the `community.postgresql.postgresql_script` module to execute statements from scripts (https://github.com/ansible-collections/community.postgresql/issues/189).
- postgresql_query - the default value of the `as_single_query` option changes to `yes`. If the related behavior of your tasks where the module is involved changes, please adjust the parameter's value correspondingly (https://github.com/ansible-collections/community.postgresql/issues/85).
- postgresql_user - the `priv` argument has been deprecated and will be removed in `community.postgresql 3.0.0`. Please use the `postgresql_privs` module to grant/revoke privileges instead (https://github.com/ansible-collections/community.postgresql/issues/212).

community.vmware

- Drop VCSIM as a test target (https://github.com/ansible-collections/community.vmware/pull/1294).

containers.podman

- Add podman_tag module
- Add secrets driver and driver opts support

dellemc.openmanage

- All modules can read custom or organizational CA signed certificate from the environment variables. Please refer to SSL Certificate Validation section in the README.md for modification to existing playbooks or setting environment variable.
- All modules now support SSL over HTTPS and socket level timeout.
- idrac_server_config_profile - The module is enhanced to support export, import, and preview the SCP configuration using Redfish and added support for check mode.

f5networks.f5_modules

- bigip_device_info - pagination logic has also been added to help with api stability.
- bigip_device_info - the module no longer gathers information from all partitions on device. This change will stabalize the module by gathering resources only from the given partition and prevent the module from gathering way too much information that might result in crashing.

fortinet.fortios

- Support FortiOS 7.0.2, 7.0.3, 7.0.4, 7.0.5.

frr.frr

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.

ibm.qradar

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.

junipernetworks.junos

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.
- *junos_facts* - change default gather_subset to *min* from *!config*.

ovirt.ovirt

- manageiq - role removed (https://github.com/oVirt/ovirt-ansible-collection/pull/375).

splunk.es

- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.

vyos.vyos

- Add 'pool' as value to server key in ntp_global.
- Minimum required ansible.netcommon version is 2.5.1.
- Updated base plugin references to ansible.netcommon.
- *vyos_facts* - change default gather_subset to *min* from *!config* (https://github.com/ansible-collections/vyos.vyos/issues/231).

Removed Collections

- community.kubernetes (previously included version: 2.0.1)
- community.kubevirt (previously included version: 1.0.0)

Removed Features

- The community.kubernetes collection has been removed from Ansible 6. It has been deprecated since Ansible 4.2, and version 2.0.0 included since Ansible 5 is only a set of deprecated redirects from community.kubernetes to kubernetes.core. If you still need the redirects, you can manually install community.kubernetes with `ansible-galaxy collection install community.kubernetes` (https://github.com/ansible-community/community-topics/issues/93).
- The community.kubevirt collection has been removed from Ansible 6. It has not been working with the community.kubernetes collection included since Ansible 5.0.0, and unfortunately nobody managed to adjust the collection to work with kubernetes.core >= 2.0.0. If you need to use this collection, you need to manually install community.kubernetes < 2.0.0 together with community.kubevirt with `ansible-galaxy collection install community.kubevirt 'community.kubernetes:<2.0.0'` (https://github.com/ansible-community/community-topics/issues/92).

Ansible-core

- Remove deprecated `Templar.set_available_variables()` method (https://github.com/ansible/ansible/issues/75828)
- cli - remove deprecated ability to set verbosity before the sub-command (https://github.com/ansible/ansible/issues/75823)
- copy - remove deprecated `thirsty` alias (https://github.com/ansible/ansible/issues/75824)
- psrp - Removed fallback on `put_file` with older `pypsrp` versions. Users must have at least `pypsrp>=0.4.0`.

- url_argument_spec - remove deprecated `thirsty` alias for `get_url` and `uri` modules (https://github.com/ansible/ansible/issues/75825, https://github.com/ansible/ansible/issues/75826)

community.general

- ali_instance_info - removed the options `availability_zone`, `instance_ids`, and `instance_names`. Use filter item `zone_id` instead of `availability_zone`, filter item `instance_ids` instead of `instance_ids`, and filter item `instance_name` instead of `instance_names` (https://github.com/ansible-collections/community.general/pull/4516).
- apt_rpm - removed the deprecated alias `update-cache` of `update_cache` (https://github.com/ansible-collections/community.general/pull/4516).
- compose - removed various deprecated aliases. Use the version with `_` instead of `-` instead (https://github.com/ansible-collections/community.general/pull/4516).
- dnsimple - remove support for dnsimple < 2.0.0 (https://github.com/ansible-collections/community.general/pull/4516).
- github_deploy_key - removed the deprecated alias `2fa_token` of `otp` (https://github.com/ansible-collections/community.general/pull/4516).
- homebrew, homebrew_cask - removed the deprecated alias `update-brew` of `update_brew` (https://github.com/ansible-collections/community.general/pull/4516).
- linode - removed the `backupsenabled` option. Use `backupweeklyday` or `backupwindow` to enable backups (https://github.com/ansible-collections/community.general/pull/4516).
- opkg - removed the deprecated alias `update-cache` of `update_cache` (https://github.com/ansible-collections/community.general/pull/4516).
- pacman - if `update_cache=true` is used with `name` or `upgrade`, the changed state will now also indicate if only the cache was updated. To keep the old behavior - only indicate `changed` when a package was installed/upgraded -, use `changed_when` as indicated in the module examples (https://github.com/ansible-collections/community.general/pull/4516).
- pacman - removed the deprecated alias `update-cache` of `update_cache` (https://github.com/ansible-collections/community.general/pull/4516).
- proxmox, proxmox_kvm, proxmox_snap - no longer allow to specify a VM name that matches multiple VMs. If this happens, the modules now fail (https://github.com/ansible-collections/community.general/pull/4516).
- serverless - removed the `functions` option. It was not used by the module (https://github.com/ansible-collections/community.general/pull/4516).
- slackpkg - removed the deprecated alias `update-cache` of `update_cache` (https://github.com/ansible-collections/community.general/pull/4516).
- urpmi - removed the deprecated alias `no-recommends` of `no_recommends` (https://github.com/ansible-collections/community.general/pull/4516).
- urpmi - removed the deprecated alias `update-cache` of `update_cache` (https://github.com/ansible-collections/community.general/pull/4516).
- xbps - removed the deprecated alias `update-cache` of `update_cache` (https://github.com/ansible-collections/community.general/pull/4516).
- xfconf - the `get` state has been removed. Use the `xfconf_info` module instead (https://github.com/ansible-collections/community.general/pull/4516).

community.hashi_vault

- aws_iam auth - the deprecated alias `aws_iam_login` for the `aws_iam` value of the `auth_method` option has been removed (https://github.com/ansible-collections/community.hashi_vault/issues/194).
- community.hashi_vault collection - support for Ansible 2.9 and ansible-base 2.10 has been removed (https://github.com/ansible-collections/community.hashi_vault/issues/189).
- hashi_vault lookup - the deprecated `[lookup_hashi_vault]` INI config section has been removed in favor of the collection-wide `[hashi_vault_collection]` section (https://github.com/ansible-collections/community.hashi_vault/issues/179).
- the "legacy" integration test setup has been removed; this does not affect end users and is only relevant to contributors (https://github.com/ansible-collections/community.hashi_vault/pull/191).

community.network

- aireos modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- aireos modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- aruba modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- aruba modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ce modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ce modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- enos modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- enos modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ironware modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- ironware modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- sros modules - removed deprecated `connection: local` support. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).
- sros modules - removed deprecated `provider` option. Use `connection: network_cli` instead (https://github.com/ansible-collections/community.network/pull/440).

community.vmware

- vcenter_extension_facts - The deprecated module `vcenter_extension_facts` has been removed, use `vcenter_extension_info` instead.
- vmware_about_facts - The deprecated module `vmware_about_facts` has been removed, use `vmware_about_info` instead.
- vmware_category_facts - The deprecated module `vmware_category_facts` has been removed, use `vmware_category_info` instead.
- vmware_cluster - Remove DRS configuration in favour of module `vmware_cluster_drs`.
- vmware_cluster - Remove HA configuration in favour of module `vmware_cluster_ha`.
- vmware_cluster - Remove VSAN configuration in favour of module `vmware_cluster_vsan`.
- vmware_cluster_facts - The deprecated module `vmware_cluster_facts` has been removed, use `vmware_cluster_info` instead.
- vmware_datastore_facts - The deprecated module `vmware_datastore_facts` has been removed, use `vmware_datastore_info` instead.
- vmware_drs_group_facts - The deprecated module `vmware_drs_group_facts` has been removed, use `vmware_drs_group_info` instead.
- vmware_drs_rule_facts - The deprecated module `vmware_drs_rule_facts` has been removed, use `vmware_drs_rule_info` instead.
- vmware_dvs_portgroup - The deprecated parameter `portgroup_type` has been removed, use `port_binding` instead.
- vmware_dvs_portgroup_facts - The deprecated module `vmware_dvs_portgroup_facts` has been removed, use `vmware_dvs_portgroup_info` instead.
- vmware_guest_boot_facts - The deprecated module `vmware_guest_boot_facts` has been removed, use `vmware_guest_boot_info` instead.
- vmware_guest_customization_facts - The deprecated module `vmware_guest_customization_facts` has been removed, use `vmware_guest_customization_info` instead.
- vmware_guest_disk_facts - The deprecated module `vmware_guest_disk_facts` has been removed, use `vmware_guest_disk_info` instead.
- vmware_guest_facts - The deprecated module `vmware_guest_facts` has been removed, use `vmware_guest_info` instead.
- vmware_guest_snapshot_facts - The deprecated module `vmware_guest_snapshot_facts` has been removed, use `vmware_guest_snapshot_info` instead.
- vmware_host_capability_facts - The deprecated module `vmware_host_capability_facts` has been removed, use `vmware_host_capability_info` instead.
- vmware_host_config_facts - The deprecated module `vmware_host_config_facts` has been removed, use `vmware_host_config_info` instead.
- vmware_host_dns_facts - The deprecated module `vmware_host_dns_facts` has been removed, use `vmware_host_dns_info` instead.
- vmware_host_feature_facts - The deprecated module `vmware_host_feature_facts` has been removed, use `vmware_host_feature_info` instead.
- vmware_host_firewall_facts - The deprecated module `vmware_host_firewall_facts` has been removed, use `vmware_host_firewall_info` instead.

- vmware_host_ntp_facts - The deprecated module `vmware_host_ntp_facts` has been removed, use `vmware_host_ntp_info` instead.
- vmware_host_package_facts - The deprecated module `vmware_host_package_facts` has been removed, use `vmware_host_package_info` instead.
- vmware_host_service_facts - The deprecated module `vmware_host_service_facts` has been removed, use `vmware_host_service_info` instead.
- vmware_host_ssl_facts - The deprecated module `vmware_host_ssl_facts` has been removed, use `vmware_host_ssl_info` instead.
- vmware_host_vmhba_facts - The deprecated module `vmware_host_vmhba_facts` has been removed, use `vmware_host_vmhba_info` instead.
- vmware_host_vmnic_facts - The deprecated module `vmware_host_vmnic_facts` has been removed, use `vmware_host_vmnic_info` instead.
- vmware_local_role_facts - The deprecated module `vmware_local_role_facts` has been removed, use `vmware_local_role_info` instead.
- vmware_local_user_facts - The deprecated module `vmware_local_user_facts` has been removed, use `vmware_local_user_info` instead.
- vmware_portgroup_facts - The deprecated module `vmware_portgroup_facts` has been removed, use `vmware_portgroup_info` instead.
- vmware_resource_pool_facts - The deprecated module `vmware_resource_pool_facts` has been removed, use `vmware_resource_pool_info` instead.
- vmware_tag_facts - The deprecated module `vmware_tag_facts` has been removed, use `vmware_tag_info` instead.
- vmware_target_canonical_facts - The deprecated module `vmware_target_canonical_facts` has been removed, use `vmware_target_canonical_info` instead.
- vmware_vm_facts - The deprecated module `vmware_vm_facts` has been removed, use `vmware_vm_info` instead.
- vmware_vmkernel_facts - The deprecated module `vmware_vmkernel_facts` has been removed, use `vmware_vmkernel_info` instead.
- vmware_vmkernel_ip_config - The deprecated module `vmware_vmkernel_ip_config` has been removed, use `vmware_vmkernel` instead.
- vmware_vswitch_facts - The deprecated module `vmware_vswitch_facts` has been removed, use `vmware_vswitch_info` instead.

Deprecated Features

- The collection `community.sap` has been renamed to `community.sap_libs`. For now both collections are included in Ansible. The content in `community.sap` will be replaced with deprecated redirects to the new collection in Ansible 7.0.0, and these redirects will eventually be removed from Ansible. Please update your FQCNs for `community.sap`.

Ansible-core

- ansible-core - Remove support for Python 2.6.
- ansible-test - Remove support for Python 2.6.
- ssh connection plugin option scp_if_ssh in favor of ssh_transfer_method.

amazon.aws

- ec2_instance - The default value for ``instance_type`` has been deprecated, in the future release you must set an instance_type or a launch_template (https://github.com/ansible-collections/amazon.aws/pull/587).
- module_utils - support for the original AWS SDK *boto* has been deprecated in favour of the *boto3/botocore* SDK. All *boto* based modules have either been deprecated or migrated to *botocore*, and the remaining support code in module_utils will be removed in release 4.0.0 of the amazon.aws collection. Any modules outside of the amazon.aws and community.aws collections based on the *boto* library will need to be migrated to the *boto3/botocore* libraries (https://github.com/ansible-collections/amazon.aws/pull/575).

cisco.ios

- Deprecates lldp module.
- ios_acls - Deprecated fragment attribute added boolean alternate as enable_fragment.

cisco.nxos

- Deprecated nxos_snmp_community module.
- Deprecated nxos_snmp_contact module.
- Deprecated nxos_snmp_host module.
- Deprecated nxos_snmp_location module.
- Deprecated nxos_snmp_traps module.
- Deprecated nxos_snmp_user module.

community.docker

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.docker 3.0.0). Some modules might still work with these versions afterwards, but we will no longer keep compatibility code that was needed to support them (https://github.com/ansible-collections/community.docker/pull/361).
- The dependency on docker-compose for Execution Environments is deprecated and will be removed in community.docker 3.0.0. The Python docker-compose library is unmaintained and can cause dependency issues. You can manually still install it in an Execution Environment when needed (https://github.com/ansible-collections/community.docker/pull/373).
- Various modules - the default of `tls_hostname` that was supposed to be removed in community.docker 2.0.0 will now be removed in version 3.0.0 (https://github.com/ansible-collections/community.docker/pull/362).
- docker_stack - the return values `out` and `err` that were supposed to be removed in community.docker 2.0.0 will now be removed in version 3.0.0 (https://github.com/ansible-collections/community.docker/pull/362).

community.general

- ansible_galaxy_install - deprecated support for `ansible` 2.9 and `ansible-base` 2.10 (https://github.com/ansible-collections/community.general/pull/4601).
- dig lookup plugin - the `DLV` record type has been decommissioned in 2017 and support for it will be removed from community.general 6.0.0 (https://github.com/ansible-collections/community.general/pull/4618).
- gem - the default of the `norc` option has been deprecated and will change to `true` in community.general 6.0.0. Explicitly specify a value to avoid a deprecation warning (https://github.com/ansible-collections/community.general/pull/4517).
- mail callback plugin - not specifying `sender` is deprecated and will be disallowed in community.general 6.0.0 (https://github.com/ansible-collections/community.general/pull/4140).
- module_helper module utils - deprecated the attribute `ModuleHelper.VarDict` (https://github.com/ansible-collections/community.general/pull/3801).
- nmcli - deprecate default hairpin mode for a bridge. This so we can change it to `false` in community.general 7.0.0, as this is also the default in `nmcli` (https://github.com/ansible-collections/community.general/pull/4334).
- pacman - from community.general 5.0.0 on, the `changed` status of `update_cache` will no longer be ignored if `name` or `upgrade` is specified. To keep the old behavior, add something like `register: result` and `changed_when: result.packages | length > 0` to your task (https://github.com/ansible-collections/community.general/pull/4329).
- proxmox inventory plugin - the current default `true` of the `want_proxmox_nodes_ansible_host` option has been deprecated. The default will change to `false` in community.general 6.0.0. To keep the current behavior, explicitly set `want_proxmox_nodes_ansible_host` to `true` in your inventory configuration. We suggest to already switch to the new behavior by explicitly setting it to `false`, and by using `compose:` to set `ansible_host` to the correct value. See the examples in the plugin documentation for details (https://github.com/ansible-collections/community.general/pull/4466).
- vmadm - deprecated module parameter `debug` that was not used anywhere (https://github.com/ansible-collections/community.general/pull/4580).

community.hashi_vault

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.hashi_vault 3.0.0) next spring (https://github.com/ansible-community/community-topics/issues/50, https://github.com/ansible-collections/community.hashi_vault/issues/189).
- aws_iam_login auth method - the `aws_iam_login` method has been renamed to `aws_iam`. The old name will be removed in collection version `3.0.0`. Until then both names will work, and a warning will be displayed when using the old name (https://github.com/ansible-collections/community.hashi_vault/pull/193).
- token_validate options - the shared auth option `token_validate` will change its default from `True` to `False` in community.hashi_vault version 4.0.0. The `vault_login` lookup and module will keep the default value of `True` (https://github.com/ansible-collections/community.hashi_vault/issues/248).
- token_validate options - the shared auth option `token_validate` will change its default from `true` to `false` in community.hashi_vault version 4.0.0. The `vault_login` lookup and module will keep the default value of `true` (https://github.com/ansible-collections/community.hashi_vault/issues/248).

community.network

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.network 4.0.0) this spring. While most content will probably still work with ansible-base 2.10, we will remove symbolic links for modules and action plugins, which will make it impossible to use them with Ansible 2.9 anymore. Please use community.network 3.x.y with Ansible 2.9 and ansible-base 2.10, as these releases will continue to support Ansible 2.9 and ansible-base 2.10 even after they are End of Life (https://github.com/ansible-community/community-topics/issues/50, https://github.com/ansible-collections/community.network/pull/382).

junipernetworks.junos

- 'router_id' options is deprecated from junos_ospf_interfaces, junos_ospfv2 and junos_ospfv3 resource module.

purestorage.flasharray

- purefa_sso - Deprecated in favor of M(purefa_admin). Will be removed in Collection 2.0

1.3.4 Ansible 5 Porting Guide

Ansible 5 is based on Ansible-core 2.12.

We suggest you read this page along with the Ansible 5 Changelog to understand what updates you may need to make.

Playbook

- When calling tasks and setting `async`, setting `ANSIBLE_ASYNC_DIR` under `environment:` is no longer valid. Instead, use the shell configuration variable `async_dir`, for example by setting `ansible_async_dir`:

```
tasks:
  - dnf:
      name: '*'
      state: latest
    async: 300
    poll: 5
    vars:
      ansible_async_dir: /path/to/my/custom/dir
```

- The `undef()` function is added to the templating environment for creating undefined variables directly in a template. Optionally, a hint may be provided for variables which are intended to be overridden.

```
vars:
  old: "{{ undef }}"
  new: "{{ undef() }}"
  new_with_hint: "{{ undef(hint='You must override this variable') }}"
```

Python Interpreter Discovery

The default value of `INTERPRETER_PYTHON` changed to `auto`. The list of Python interpreters in `INTERPRETER_PYTHON_FALLBACK` changed to prefer Python 3 over Python 2. The combination of these two changes means the new default behavior is to quietly prefer Python 3 over Python 2 on remote hosts. Previously a deprecation warning was issued in situations where interpreter discovery would have used Python 3 but the interpreter was set to `/usr/bin/python`.

`INTERPRETER_PYTHON_FALLBACK` can be changed from the default list of interpreters by setting the `ansible_interpreter_python_fallback` variable.

See *interpreter discovery documentation* for more details.

Command Line

- Python 3.8 on the controller node is a hard requirement for this release. The command line scripts will not function with a lower Python version.
- `ansible-vault` no longer supports `PyCrypto` and requires `cryptography`.

Deprecated

- Python 2.6 on the target node is deprecated in this release. `ansible-core` 2.13 will remove support for Python 2.6.
- Bare variables in conditionals: `when` conditionals no longer automatically parse string booleans such as `"true"` and `"false"` into actual booleans. Any variable containing a non-empty string is considered true. This was previously configurable with the `CONDITIONAL_BARE_VARS` configuration option (and the `ANSIBLE_CONDITIONAL_BARE_VARS` environment variable). This setting no longer has any effect. Users can work around the issue by using the `|bool` filter:

```
vars:
  teardown: 'false'

tasks:
  - include_tasks: teardown.yml
    when: teardown | bool

  - include_tasks: provision.yml
    when: not teardown | bool
```

- The `_remote_checksum()` method in `ActionBase` is deprecated. Any action plugin using this method should use `_execute_remote_stat()` instead.

Modules

- `cron` now requires `name` to be specified in all cases.
- `cron` no longer allows a `reboot` parameter. Use `special_time: reboot` instead.
- `hostname` - On FreeBSD, the `before` result will no longer be `"temporarystub"` if permanent hostname file does not exist. It will instead be `""` (empty string) for consistency with other systems.
- `hostname` - On OpenRC and Solaris based systems, the `before` result will no longer be `"UNKNOWN"` if the permanent hostname file does not exist. It will instead be `""` (empty string) for consistency with other systems.
- `pip` now uses the `pip` Python module installed for the Ansible module's Python interpreter, if available, unless `executable` or `virtualenv` were specified.

Modules removed

The following modules no longer exist:

- No notable changes

Deprecation notices

No notable changes

Noteworthy module changes

No notable changes

Plugins

- `unique` filter with Jinja2 < 2.10 is case-sensitive and now raise coherently an error if `case_sensitive=False` instead of when `case_sensitive=True`.
- Set theory filters (`intersect`, `difference`, `symmetric_difference` and `union`) are now case-sensitive. Explicitly use `case_sensitive=False` to keep previous behavior. Note: with Jinja2 < 2.10, the filters were already case-sensitive by default.
- `password_hash`` now uses `passlib` defaults when an option is unspecified, for example `bcrypt_sha256`, now default to the "2b" format and if the "2a" format is required it must be specified.

Porting custom scripts

No notable changes

Networking

No notable changes

Porting Guide for v5.9.0

Added Collections

- cisco.dnac (version 6.4.0)
- community.sap_libs (version 1.1.0)

Major Changes

fortinet.fortios

- Support FortiOS 7.0.2, 7.0.3, 7.0.4, 7.0.5.

Deprecated Features

- The collection `community.sap` has been renamed to `community.sap_libs`. For now both collections are included in Ansible. The content in `community.sap` will be replaced with deprecated redirects to the new collection in Ansible 7.0.0, and these redirects will eventually be removed from Ansible. Please update your FQCNs for `community.sap`.

community.docker

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.docker 3.0.0). Some modules might still work with these versions afterwards, but we will no longer keep compatibility code that was needed to support them (https://github.com/ansible-collections/community.docker/pull/361).
- The dependency on docker-compose for Execution Environments is deprecated and will be removed in community.docker 3.0.0. The Python docker-compose library is unmaintained and can cause dependency issues. You can manually still install it in an Execution Environment when needed (https://github.com/ansible-collections/community.docker/pull/373).
- Various modules - the default of `tls_hostname` that was supposed to be removed in community.docker 2.0.0 will now be removed in version 3.0.0 (https://github.com/ansible-collections/community.docker/pull/362).
- docker_stack - the return values `out` and `err` that were supposed to be removed in community.docker 2.0.0 will now be removed in version 3.0.0 (https://github.com/ansible-collections/community.docker/pull/362).

Porting Guide for v5.8.0

Added Collections

- vmware.vmware_rest (version 2.1.5)

Breaking Changes

vmware.vmware_rest

- The vmware_rest 2.0.0 support vSphere 7.0.2 onwards.
- vcenter_vm_storage_policy - the format of the `disks` parameter has changed.
- vcenter_vm_storage_policy - the module has a new mandatory parameter: `vm_home`.

Major Changes

community.mysql

- The community.mysql collection no longer supports `Ansible 2.9` and `ansible-base 2.10`. While we take no active measures to prevent usage and there are no plans to introduce incompatible code to the modules, we will stop testing against `Ansible 2.9` and `ansible-base 2.10`. Both will very soon be End of Life and if you are still using them, you should consider upgrading to the `latest Ansible / ansible-core 2.11 or later` as soon as possible (https://github.com/ansible-collections/community.mysql/pull/343).

community.postgresql

- The community.postgresql collection no longer supports `Ansible 2.9` and `ansible-base 2.10`. While we take no active measures to prevent usage and there are no plans to introduce incompatible code to the modules, we will stop testing against `Ansible 2.9` and `ansible-base 2.10`. Both will very soon be End of Life and if you are still using them, you should consider upgrading to the `latest Ansible / ansible-core 2.11 or later` as soon as possible (https://github.com/ansible-collections/community.postgresql/pull/245).

Deprecated Features

community.hashi_vault

- token_validate options - the shared auth option `token_validate` will change its default from `True` to `False` in community.hashi_vault version 4.0.0. The `vault_login` lookup and module will keep the default value of `True` (https://github.com/ansible-collections/community.hashi_vault/issues/248).

community.network

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.network 4.0.0) this spring. While most content will probably still work with ansible-base 2.10, we will remove symbolic links for modules and action plugins, which will make it impossible to use them with Ansible 2.9 anymore. Please use community.network 3.x.y with Ansible 2.9 and ansible-base 2.10, as these releases will continue to support Ansible 2.9 and ansible-base 2.10 even after they are End of Life (https://github.com/ansible-community/community-topics/issues/50, https://github.com/ansible-collections/community.network/pull/382).

vmware.vmware_rest

- vcenter_vm_storage_policy_compliance - drop the module, it returns 404 error.
- vcenter_vm_tools - remove the `upgrade` state.
- vcenter_vm_tools_installer - remove the module from the collection.

Porting Guide for v5.7.0

Major Changes

community.postgresql

- postgresql_user - the `priv` argument has been deprecated and will be removed in `community.postgresql 3.0.0`. Please use the `postgresql_privs` module to grant/revoke privileges instead (https://github.com/ansible-collections/community.postgresql/issues/212).

fortinet.fortios

- Support FortiOS 7.0.2, 7.0.3, 7.0.4, 7.0.5.

Deprecated Features

community.general

- nmcli - deprecate default hairpin mode for a bridge. This so we can change it to `false` in community.general 7.0.0, as this is also the default in `nmcli` (https://github.com/ansible-collections/community.general/pull/4334).
- proxmox inventory plugin - the current default `true` of the `want_proxmox_nodes_ansible_host` option has been deprecated. The default will change to `false` in community.general 6.0.0. To keep the current behavior, explicitly set `want_proxmox_nodes_ansible_host` to `true` in your inventory configuration. We suggest to already switch to the new behavior by explicitly setting it to `false`, and by using `compose:` to set `ansible_host` to the correct value. See the examples in the plugin documentation for details (https://github.com/ansible-collections/community.general/pull/4466).

Porting Guide for v5.6.0

Added Collections

- community.sap (version 1.0.0)

Deprecated Features

cisco.ios

- Deprecates lldp module.

Porting Guide for v5.5.0

Known Issues

community.general

- pacman - `update_cache` cannot differentiate between up to date and outdated package lists and will report `changed` in both situations (https://github.com/ansible-collections/community.general/pull/4318).
- pacman - binaries specified in the `executable` parameter must support `--print-format` in order to be used by this module. In particular, AUR helper `yay` is known not to currently support it (https://github.com/ansible-collections/community.general/pull/4312).

Deprecated Features

community.general

- pacman - from community.general 5.0.0 on, the `changed` status of `update_cache` will no longer be ignored if `name` or `upgrade` is specified. To keep the old behavior, add something like `register: result` and `changed_when: result.packages | length > 0` to your task (https://github.com/ansible-collections/community.general/pull/4329).

Porting Guide for v5.4.0

Major Changes

chocolatey.chocolatey

- win_chocolatey - Added choco_args option to pass additional arguments directly to Chocolatey.

vyos.vyos

- Add 'pool' as value to server key in ntp_global.

Deprecated Features

cisco.ios

- *ios_acls* - Deprecated fragment attribute added boolean alternate as enable_fragment.

Porting Guide for v5.3.0

Major Changes

f5networks.f5_modules

- bigip_device_info - pagination logic has also been added to help with api stability.
- bigip_device_info - the module no longer gathers information from all partitions on device. This change will stabalize the module by gathering resources only from the given partition and prevent the module from gathering way too much information that might result in crashing.

Deprecated Features

community.general

- mail callback plugin - not specifying `sender` is deprecated and will be disallowed in community.general 6.0.0 (https://github.com/ansible-collections/community.general/pull/4140).

Porting Guide for v5.2.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) The module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_application_alerts_smtp - Issue(212310) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_application_alerts_syslog - Issue(215374) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_device_local_access_configuration - Issue(215035) - The module reports `Successfully updated the local access setting` if an unsupported value is provided for the parameter timeout_limit. However, this value is not actually applied on OpenManage Enterprise Modular.
- ome_device_local_access_configuration - Issue(217865) - The module does not display a proper error message if an unsupported value is provided for the user_defined and lcd_language parameters.
- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_power_settings - Issue(212679) - The module errors out with the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_smart_fabric_uplink - Issue(186024) - The module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

purestorage.flasharray

- purefa_admin - Once *max_login* and *lockout* have been set there is currently no way to rest these to zero except through the FlashArray GUI

Major Changes

cisco.meraki

- meraki_mr_radio - New module

Deprecated Features

purestorage.flasharray

- purefa_sso - Deprecated in favor of M(purefa_admin). Will be removed in Collection 2.0

Porting Guide for v5.1.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) The module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_application_alerts_smtp - Issue(212310) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_application_alerts_syslog - Issue(215374) - The module does not provide a proper error message if the destination_address is more than 255 characters.
- ome_device_network_services - Issue(212681) - The module does not provide a proper error message if unsupported values are provided for the parameters- port_number, community_name, max_sessions, max_auth_retries, and idle_timeout.
- ome_device_power_settings - Issue(212679) - The module errors out with the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_smart_fabric_uplink - Issue(186024) - The module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Major Changes

containers.podman

- Add podman_tag module
- Add secrets driver and driver opts support

Removed Features

community.hashi_vault

- the "legacy" integration test setup has been removed; this does not affect end users and is only relevant to contributors (https://github.com/ansible-collections/community.hashi_vault/pull/191).

Deprecated Features

cisco.nxos

- Deprecated nxos_snmp_community module.
- Deprecated nxos_snmp_contact module.
- Deprecated nxos_snmp_host module.
- Deprecated nxos_snmp_location module.
- Deprecated nxos_snmp_traps module.
- Deprecated nxos_snmp_user module.

community.general

- module_helper module utils - deprecated the attribute `ModuleHelper.VarDict` (https://github.com/ansible-collections/community.general/pull/3801).

community.hashi_vault

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.hashi_vault 3.0.0) next spring (https://github.com/ansible-community/community-topics/issues/50, https://github.com/ansible-collections/community.hashi_vault/issues/189).
- aws_iam_login auth method - the `aws_iam_login` method has been renamed to `aws_iam`. The old name will be removed in collection version `3.0.0`. Until then both names will work, and a warning will be displayed when using the old name (https://github.com/ansible-collections/community.hashi_vault/pull/193).

junipernetworks.junos

- 'router_id' options is deprecated from junos_ospf_interfaces, junos_ospfv2 and junos_ospfv3 resource module.

Porting Guide for v5.0.1

Major Changes

- Raised python requirement of the ansible package from >=2.7 to >=3.8 to match ansible-core

Porting Guide for v5.0.0

Added Collections

- cisco.ise (version 1.2.1)
- cloud.common (version 2.1.0)
- community.ciscosmb (version 1.0.4)
- community.dns (version 2.0.3)
- infoblox.nios_modules (version 1.1.2)
- netapp.storagegrid (version 21.7.0)

Known Issues

Ansible-core

- ansible-test - Tab completion anywhere other than the end of the command with the new composite options will provide incorrect results. See issue 351 for additional details.

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_device_power_settings - Issue(212679) The ome_device_power_settings module errors out with the following message if the value provided for the parameter `power_cap` is not within the supported range of 0 to 32767, `Unable to complete the request because PowerCap does not exist or is not applicable for the resource URI.`
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though it is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

purestorage.flashblade

- purefb_lag - The mac_address field in the response is not populated. This will be fixed in a future FlashBlade update.

Breaking Changes

Ansible-core

- Action, module, and group names in module_defaults must be static values. Their values can still be templates.
- Fully qualified 'ansible.legacy' plugin names are not included implicitly in action_groups.
- Unresolvable groups, action plugins, and modules in module_defaults are an error.
- ansible-test - Automatic installation of requirements for "cloud" test plugins no longer occurs. The affected test plugins are `aws`, `azure`, `cs`, `hcloud`, `nios`, `opennebula`, `openshift` and `vcenter`. Collections should instead use one of the supported integration test requirements files, such as the `tests/integration/requirements.txt` file.
- ansible-test - The HTTP Tester is no longer available with the `ansible-test shell` command. Only the `integration` and `windows-integration` commands provide HTTP Tester.
- ansible-test - The `--disable-httptester` option is no longer available. The HTTP Tester is no longer optional for tests that specify it.
- ansible-test - The `--httptester` option is no longer available. To override the container used for HTTP Tester tests, set the `ANSIBLE_HTTP_TEST_CONTAINER` environment variable instead.
- ansible-test - Unit tests for `modules` and `module_utils` are now limited to importing only `ansible.module_utils` from the `ansible` module.
- conditionals - `when` conditionals no longer automatically parse string booleans such as `"true"` and `"false"` into actual booleans. Any non-empty string is now considered true. The `CONDITIONAL_BARE_VARS` configuration variable no longer has any effect.
- hostname - Drops any remaining support for Python 2.4 by using `with open()` to simplify exception handling code which leaked file handles in several spots
- hostname - On FreeBSD, the string `temporarystub` no longer gets written to the hostname file in the get methods (and in check_mode). As a result, the default hostname will now appear as `''` (empty string) instead of `temporarystub` for consistency with other strategies. This means the `before` result will be different.
- hostname - On OpenRC systems and Solaris, the `before` value will now be `''` (empty string) if the permanent hostname file does not exist, for consistency with other strategies.
- intersect, difference, symmetric_difference, union filters - the default behavior is now to be case-sensitive (https://github.com/ansible/ansible/issues/74255)
- unique filter - the default behavior is now to fail if Jinja2's filter fails and explicit `case_sensitive=False` as the Ansible's fallback is case-sensitive (https://github.com/ansible/ansible/pull/74256)

amazon.aws

- ec2_instance - instance wait for state behaviour has changed. If plays require the old behavior of waiting for the instance monitoring status to become `OK` when launching a new instance, the action will need to specify `state: started` (https://github.com/ansible-collections/amazon.aws/pull/481).
- ec2_snapshot - support for waiting indefinitely has been dropped, new default is 10 minutes (https://github.com/ansible-collections/amazon.aws/pull/356).
- ec2_vol_info - return `attachment_set` is now a list of attachments with Multi-Attach support on disk. (https://github.com/ansible-collections/amazon.aws/pull/362).
- ec2_vpc_dhcp_option - The module has been refactored to use boto3. Keys and value types returned by the module are now consistent, which is a change from the previous behaviour. A `purge_tags` option has been added, which defaults to `True`. (https://github.com/ansible-collections/amazon.aws/pull/252)
- ec2_vpc_dhcp_option_info - Now preserves case for tag keys in return value. (https://github.com/ansible-collections/amazon.aws/pull/252)
- module_utils.core - The boto3 switch has been removed from the region parameter (https://github.com/ansible-collections/amazon.aws/pull/287).
- module_utils/compat - vendored copy of ipaddress removed (https://github.com/ansible-collections/amazon.aws/pull/461).
- module_utils/core - updated the `scrub_none_parameters` function so that `descend_into_lists` is set to `True` by default (https://github.com/ansible-collections/amazon.aws/pull/297).

arista.eos

- Arista released train 4.23.X and newer and along with it replaced and deprecated lots of commands. This PR adds support for syntax changes in release train 4.23 and after. Going forward the eos modules will not support eos sw version < 4.23.

community.aws

- ec2_instance - The module has been migrated to the `amazon.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_instance`.
- ec2_instance_info - The module has been migrated to the `amazon.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_instance_info`.
- ec2_vpc_endpoint - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint`.
- ec2_vpc_endpoint_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint_info`.
- ec2_vpc_endpoint_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint_info`.
- ec2_vpc_endpoint_service_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint_service_info`.

- ec2_vpc_igw - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_igw`.
- ec2_vpc_igw_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_igw_info`.
- ec2_vpc_igw_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_igw_info`.
- ec2_vpc_nat_gateway - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_nat_gateway`.
- ec2_vpc_nat_gateway_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_nat_gateway_info`.
- ec2_vpc_nat_gateway_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_nat_gateway_info`.
- kms_info - key details are now returned in the `kms_keys` attribute rather than the `keys` attribute (https://github.com/ansible-collections/community.aws/pull/648).

community.crypto

- Adjust `dirName` text parsing and to text converting code to conform to Sections 2 and 3 of RFC 4514. This is similar to how cryptography handles this (https://github.com/ansible-collections/community.crypto/pull/274).
- acme module utils - removing compatibility code (https://github.com/ansible-collections/community.crypto/pull/290).
- acme_* modules - removed vendored copy of the Python library `ipaddress`. If you are using Python 2.x, please make sure to install the library (https://github.com/ansible-collections/community.crypto/pull/287).
- compatibility module_utils - removed vendored copy of the Python library `ipaddress` (https://github.com/ansible-collections/community.crypto/pull/287).
- crypto module utils - removing compatibility code (https://github.com/ansible-collections/community.crypto/pull/290).
- get_certificate, openssl_csr_info, x509_certificate_info - depending on the `cryptography` version used, the modules might not return the ASN.1 value for an extension as contained in the certificate respectively CSR, but a re-encoded version of it. This should usually be identical to the value contained in the source file, unless the value was malformed. For extensions not handled by C(cryptography) the value contained in the source file is always returned unaltered (https://github.com/ansible-collections/community.crypto/pull/318).
- module_utils - removed various PyOpenSSL support functions and default backend values that are not needed for the openssl_pkcs12 module (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_csr, openssl_csr_pipe, x509_crl - the `subject` respectively `issuer` fields no longer ignore empty values, but instead fail when encountering them (https://github.com/ansible-collections/community.crypto/pull/316).
- openssl_privatekey_info - by default consistency checks are not run; they need to be explicitly requested by passing `check_consistency=true` (https://github.com/ansible-collections/community.crypto/pull/309).
- x509_crl - for idempotency checks, the `issuer` order is ignored. If order is important, use the new `issuer_ordered` option (https://github.com/ansible-collections/community.crypto/pull/316).

community.dns

- All Hetzner modules and plugins which handle DNS records now work with unquoted TXT values by default. The old behavior can be obtained by setting `txt_transformation=api` (https://github.com/ansible-collections/community.dns/issues/48, https://github.com/ansible-collections/community.dns/pull/57, https://github.com/ansible-collections/community.dns/pull/60).
- Hosttech API creation - now requires a `ModuleOptionProvider` object instead of an `AnsibleModule` object. Alternatively an Ansible plugin instance can be passed (https://github.com/ansible-collections/community.dns/pull/37).
- The hetzner_dns_record_info and hosttech_dns_record_info modules have been renamed to hetzner_dns_record_set_info and hosttech_dns_record_set_info, respectively (https://github.com/ansible-collections/community.dns/pull/54).
- The hosttech_dns_record module has been renamed to hosttech_dns_record_set (https://github.com/ansible-collections/community.dns/pull/31).
- The internal bulk record updating helper (`bulk_apply_changes`) now also returns the records that were deleted, created or updated (https://github.com/ansible-collections/community.dns/pull/63).
- The internal record API no longer allows to manage comments explicitly (https://github.com/ansible-collections/community.dns/pull/63).
- When using the internal modules API, now a zone ID type and a provider information object must be passed (https://github.com/ansible-collections/community.dns/pull/27).
- hetzner_dns_record* modules - implement correct handling of default TTL. The value `none` is now accepted and returned in this case (https://github.com/ansible-collections/community.dns/pull/52, https://github.com/ansible-collections/community.dns/issues/50).
- hetzner_dns_record, hetzner_dns_record_set, hetzner_dns_record_sets - the default TTL is now 300 and no longer 3600, which equals the default in the web console (https://github.com/ansible-collections/community.dns/pull/43).
- hosttech_* module_utils - completely rewrite and refactor to support new JSON API and allow to re-use provider-independent module logic (https://github.com/ansible-collections/community.dns/pull/4).
- hosttech_dns_record_set - the option `overwrite` was replaced by a new option `on_existing`. Specifying `overwrite=true` is equivalent to `on_existing=replace` (the new default). Specifying `overwrite=false` with `state=present` is equivalent to `on_existing=keep_and_fail`, and specifying `overwrite=false` with `state=absent` is equivalent to `on_existing=keep` (https://github.com/ansible-collections/community.dns/pull/31).

community.docker

- docker_compose - fixed `timeout` defaulting behavior so that `stop_grace_period`, if defined in the compose file, will be used if *timeout*` is not specified (https://github.com/ansible-collections/community.docker/pull/163).

community.general

- archive - adding idempotency checks for changes to file names and content within the `destination` file (https://github.com/ansible-collections/community.general/pull/3075).
- lxd inventory plugin - when used with Python 2, the plugin now needs `ipaddress` installed from pypi (https://github.com/ansible-collections/community.general/pull/2441).
- scaleway_security_group_rule - when used with Python 2, the module now needs `ipaddress` installed from pypi (https://github.com/ansible-collections/community.general/pull/2441).

community.hashi_vault

- connection options - there is no longer a default value for the `url` option (the Vault address), so a value must be supplied (https://github.com/ansible-collections/community.hashi_vault/issues/83).

community.okd

- drop python 2 support (https://github.com/openshift/community.okd/pull/93).

community.routeros

- api - due to a programming error, the module never failed on errors. This has now been fixed. If you are relying on the module not failing in case of idempotent commands (resulting in errors like `failure: already have such address`), you need to adjust your roles/playbooks. We suggest to use `failed_when` to accept failure in specific circumstances, for example `failed_when: "'failure: already have ' in result.msg[0]"` (https://github.com/ansible-collections/community.routeros/pull/39).
- api - splitting commands no longer uses a naive split by whitespace, but a more RouterOS CLI compatible splitting algorithm (https://github.com/ansible-collections/community.routeros/pull/45).
- command - the module now always indicates that a change happens. If this is not correct, please use `changed_when` to determine the correct changed status for a task (https://github.com/ansible-collections/community.routeros/pull/50).

community.zabbix

- all roles now reference other roles and modules through their fully qualified collection names, which makes Ansible 2.10 minimum supported version for roles (See issue 477).

kubernetes.core

- Drop python 2 support (https://github.com/ansible-collections/kubernetes.core/pull/86).
- helm_plugin - remove unused `release_namespace` parameter (https://github.com/ansible-collections/kubernetes.core/pull/85).
- helm_plugin_info - remove unused `release_namespace` parameter (https://github.com/ansible-collections/kubernetes.core/pull/85).
- k8s_cluster_info - returned apis as list to avoid being overwritten in case of multiple version (https://github.com/ansible-collections/kubernetes.core/pull/41).

- k8s_facts - remove the deprecated alias from k8s_facts to k8s_info (https://github.com/ansible-collections/kubernetes.core/pull/125).

netapp.storagegrid

- This version introduces a breaking change. All modules have been renamed from `nac_sg_*` to `na_sg_*`. Playbooks and Roles must be updated to match.

Major Changes

Ansible-core

- Python Controller Requirement - Python 3.8 or newer is required for the control node (the machine that runs Ansible) (https://github.com/ansible/ansible/pull/74013)
- ansible-test - All "cloud" plugins which use containers can now be used with all POSIX and Windows hosts. Previously the plugins did not work with Windows at all, and support for hosts created with the `--remote` option was inconsistent.
- ansible-test - Collections can now specify controller and target specific integration test requirements and constraints. If provided, they take precedence over the previously available requirements and constraints files.
- ansible-test - Integration tests run with the `integration` command can now be executed on two separate hosts instead of always running on the controller. The target host can be one provided by `ansible-test` or by the user, as long as it is accessible using SSH.
- ansible-test - Most container features are now supported under Podman. Previously a symbolic link for `docker` pointing to `podman` was required.
- ansible-test - New `--controller` and `--target` / `--target-python` options have been added to allow more control over test environments.
- ansible-test - Python 3.8 - 3.10 are now required to run `ansible-test`, thus matching the Ansible controller Python requirements. Older Python versions (2.6 - 2.7 and 3.5 - 3.10) can still be the target for relevant tests.
- ansible-test - SSH port forwarding and redirection is now used exclusively to make container ports available on non-container hosts. When testing on POSIX systems this requires SSH login as root. Previously SSH port forwarding was combined with firewall rules or other port redirection methods, with some platforms being unsupported.
- ansible-test - Sanity tests always run in isolated Python virtual environments specific to the requirements of each test. The environments are cached.
- ansible-test - Sanity tests are now separated into two categories, controller and target. All tests except `import` and `compile` are controller tests. The controller tests always run using the same Python version used to run `ansible-test`. The target tests use the Python version(s) specified by the user, or all available Python versions.
- ansible-test - Sanity tests now use fully pinned requirements that are independent of each other and other test types.
- ansible-test - Tests run with the `centos6` and `default` test containers now use a PyPI proxy container to access PyPI when Python 2.6 is used. This allows tests running under Python 2.6 to continue functioning even though PyPI is discontinuing support for non-SNI capable clients.
- ansible-test - The `future-import-boilerplate` and `metaclass-boilerplate` sanity tests are limited to remote-only code. Additionally, they are skipped for collections which declare no support for Python 2.x.

- ansible-test - The `import` and `compile` sanity tests limit remote-only Python version checks to remote-only code.
- ansible-test - Unit tests for controller-only code now require Python 3.8 or later.
- ansible-test - Version neutral sanity tests now require Python 3.8 or later.
- junit callback - The `junit_xml` and `ordereddict` Python modules are no longer required to use the `junit` callback plugin.

amazon.aws

- amazon.aws collection - Due to the AWS SDKs announcing the end of support for Python less than 3.6 (https://boto3.amazonaws.com/v1/documentation/api/1.17.64/guide/migrationpy3.html) this collection now requires Python 3.6+ (https://github.com/ansible-collections/amazon.aws/pull/298).
- amazon.aws collection - The amazon.aws collection has dropped support for `botocore<1.18.0` and `boto3<1.15.0`. Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/amazon.aws/pull/502).
- ec2_instance - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_instance`.
- ec2_instance_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_instance_info`.
- ec2_vpc_endpoint - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint`.
- ec2_vpc_endpoint_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint_info`.
- ec2_vpc_endpoint_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint_info`.
- ec2_vpc_endpoint_service_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_endpoint_service_info`.
- ec2_vpc_igw - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_igw`.
- ec2_vpc_igw_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_igw_facts`.
- ec2_vpc_igw_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_igw_info`.
- ec2_vpc_nat_gateway - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_nat_gateway`.
- ec2_vpc_nat_gateway_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_nat_gateway_info`.

- ec2_vpc_nat_gateway_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_nat_gateway_info`.
- ec2_vpc_route_table - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_route_table`.
- ec2_vpc_route_table_facts - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_route_table_facts`.
- ec2_vpc_route_table_info - The module has been migrated from the `community.aws` collection. Playbooks using the Fully Qualified Collection Name for this module should be updated to use `amazon.aws.ec2_vpc_route_table_info`.

cisco.ise

- Adds `ise_uses_api_gateway` to module options.
- Adds a 'aws_deployment' role that allows the deployment of an arbitrary large ISE cluster to AWS.
- Adds ise_responses to return values of info modules.
- Adds ise_update_response to return values of non-info modules.
- Fixes inner logic of modules that have no get by name and have not working filter.
- Renamed module device_administration_authorization_exception_rules to device_administration_local_exception_rules.
- Renamed module device_administration_authorization_global_exception_rules to device_administration_global_exception_rules.
- Renamed module network_access_authorization_exception_rules to network_access_local_exception_rules.
- Renamed module network_access_authorization_global_exception_rules to network_access_global_exception_rules.
- Updates options required for modules.
- Updates sdk parameters for previous modules
- device_administration_authorization_exception_rules - removed module.
- device_administration_authorization_exception_rules_info - removed module.
- device_administration_authorization_global_exception_rules - removed module.
- device_administration_authorization_global_exception_rules_info - removed module.
- guest_user_reinstante - removed module.
- import_trust_cert - removed module.
- network_access_authorization_exception_rules - removed module.
- network_access_authorization_exception_rules_info - removed module.
- network_access_authorization_global_exception_rules - removed module.
- network_access_authorization_global_exception_rules_info - removed module.
- personas_check_standalone - Adds module for the deployment of personas to existing nodes in an ISE cluster.

- personas_export_certs - Adds module for the deployment of personas to existing nodes in an ISE cluster.
- personas_promote_primary - Adds module for the deployment of personas to existing nodes in an ISE cluster.
- personas_update_roles - Adds module for the deployment of personas to existing nodes in an ISE cluster.
- service_info - removed module.
- system_certificate_export - removed module.
- telemetry_info_info - removed module.

cloud.common

- turbo - enable turbo mode for lookup plugins

cloudscale_ch.cloud

- Add custom_image module

community.aws

- community.aws collection - The community.aws collection has dropped support for `botocore<1.18.0` and `boto3<1.15.0` (https://github.com/ansible-collections/community.aws/pull/711). Most modules will continue to work with older versions of the AWS SDK, however compatibility with older versions of the SDK is not guaranteed and will not be tested. When using older versions of the SDK a warning will be emitted by Ansible (https://github.com/ansible-collections/amazon.aws/pull/442).

community.ciscosmb

- Python 2.6, 2.7, 3.5 is required
- add CBS350 support
- add antsibull-changelog support
- add ciscosmb_command
- added facts subset "interfaces"
- ciscosmb_facts with default subset and unit tests
- interface name canonicalization
- transform collection qaxi.ciscosmb to community.ciscosmb
- transform community.ciscosmb.ciscosmb_command to community.ciscosmb.command
- transform community.ciscosmb.ciscosmb_facts to community.ciscosmb.facts
- unit tests for CBS350

community.dns

- hosttech_* modules - support the new JSON API at https://api.ns1.hosttech.eu/api/documentation/ (https://github.com/ansible-collections/community.dns/pull/4).

community.general

- bitbucket_* modules - `client_id` is no longer marked as `no_log=true`. If you relied on its value not showing up in logs and output, please mark the whole tasks with `no_log: true` (https://github.com/ansible-collections/community.general/pull/2045).

community.kubernetes

- redirect everything from `community.kubernetes` to `kubernetes.core` (https://github.com/ansible-collections/community.kubernetes/pull/425).

community.okd

- update to use kubernetes.core 2.0 (https://github.com/openshift/community.okd/pull/93).

community.postgresql

- postgresql_query - the default value of the `as_single_query` option will be changed to `yes` in community.postgresql 2.0.0 (https://github.com/ansible-collections/community.postgresql/issues/85).

community.vmware

- vmware_object_custom_attributes_info - added a new module to gather custom attributes of an object (https://github.com/ansible-collections/community.vmware/pull/851).

containers.podman

- Add systemd generation for pods
- Generate systemd service files for containers

dellemc.openmanage

- idrac_server_config_profile - Added support for exporting and importing Server Configuration Profile through HTTP/HTTPS share.
- ome_device_group - Added support for adding devices to a group using the IP addresses of the devices and group ID.
- ome_firmware - Added option to stage the firmware update and support for selecting components and devices for baseline-based firmware update.
- ome_firmware_baseline - Module supports check mode, and allows the modification and deletion of firmware baselines.

- ome_firmware_catalog - Module supports check mode, and allows the modification and deletion of firmware catalogs.

fortinet.fortios

- Add real-world use cases in the example section for some configuration modules.
- Collect the current configurations of the modules and convert them into playbooks.
- Improve `fortios_configuration_fact` to use multiple selectors concurrently.
- New module fortios_monitor_fact.
- Support FortiOS 7.0.1.
- Support Fortios 7.0.
- Support Log APIs.
- Support `check_mode` in all cofigurationAPI-based modules.
- Support filtering for fact gathering modules `fortios_configuration_fact` and `fortios_monitor_fact`.
- Support member operation (delete/add extra members) on an object that has a list of members in it.
- Support moving policy in `firewall_central_snat_map`.
- Support selectors feature in `fortios_monitor_fact` and `fortios_log_fact`.
- Unify schemas for monitor API.

gluster.gluster

- enable client.ssl,server.ssl before starting the gluster volume (https://github.com/gluster/gluster-ansible-collection/pull/19)

hetzner.hcloud

- Introduction of placement groups

kubernetes.core

- k8s - deprecate merge_type=json. The JSON patch functionality has never worked (https://github.com/ansible-collections/kubernetes.core/pull/99).
- k8s_json_patch - split JSON patch functionality out into a separate module (https://github.com/ansible-collections/kubernetes.core/pull/99).
- replaces the openshift client with the official kubernetes client (https://github.com/ansible-collections/kubernetes.core/issues/34).

netapp.cloudmanager

- Adding stage environment to all modules in cloudmanager

netbox.netbox

- packages is now a required Python package and gets installed through Ansible 2.10+.

openvswitch.openvswitch

- By mistake we tagged the repo to 2.0.0 and as it wasn't intended and cannot be reverted we're releasing 2.0.1 to make the community aware of the major version update.

ovirt.ovirt

- remove_stale_lun - Add role for removing stale LUN (https://bugzilla.redhat.com/1966873).

Removed Features

Ansible-core

- The built-in module_util `ansible.module_utils.common.removed` was previously deprecated and has been removed.
- connections, removed password check stubs that had been moved to become plugins.
- task, inline parameters being auto coerced into variables has been removed.

ansible.windows

- win_reboot - Removed `shutdown_timeout` and `shutdown_timeout_sec` which has not done anything since Ansible 2.5.

community.crypto

- acme_* modules - the `acme_directory` option is now required (https://github.com/ansible-collections/community.crypto/pull/290).
- acme_* modules - the `acme_version` option is now required (https://github.com/ansible-collections/community.crypto/pull/290).
- acme_account_facts - the deprecated redirect has been removed. Use community.crypto.acme_account_info instead (https://github.com/ansible-collections/community.crypto/pull/290).
- acme_account_info - `retrieve_orders=url_list` no longer returns the return value `orders`. Use the `order_uris` return value instead (https://github.com/ansible-collections/community.crypto/pull/290).
- crypto.info module utils - the deprecated redirect has been removed. Use `crypto.pem` instead (https://github.com/ansible-collections/community.crypto/pull/290).

- get_certificate - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_certificate - the deprecated redirect has been removed. Use community.crypto.x509_certificate instead (https://github.com/ansible-collections/community.crypto/pull/290).
- openssl_certificate_info - the deprecated redirect has been removed. Use community.crypto.x509_certificate_info instead (https://github.com/ansible-collections/community.crypto/pull/290).
- openssl_csr - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_csr and openssl_csr_pipe - `version` now only accepts the (default) value 1 (https://github.com/ansible-collections/community.crypto/pull/290).
- openssl_csr_info - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_csr_pipe - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_privatekey - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_privatekey_info - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_privatekey_pipe - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_publickey - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_publickey_info - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_signature - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- openssl_signature_info - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- x509_certificate - remove `assertonly` provider (https://github.com/ansible-collections/community.crypto/pull/289).
- x509_certificate - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- x509_certificate_info - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).
- x509_certificate_pipe - removed the `pyopenssl` backend (https://github.com/ansible-collections/community.crypto/pull/273).

community.docker

- docker_container - the default value of `container_default_behavior` changed to `no_defaults` (https://github.com/ansible-collections/community.docker/pull/210).
- docker_container - the default value of `network_mode` is now the name of the first network specified in `networks` if such are specified and `networks_cli_compatible=true` (https://github.com/ansible-collections/community.docker/pull/210).
- docker_container - the special value `all` can no longer be used in `published_ports` next to other values. Please use `publish_all_ports=true` instead (https://github.com/ansible-collections/community.docker/pull/210).
- docker_login - removed the `email` option (https://github.com/ansible-collections/community.docker/pull/210).

community.general

- All inventory and vault scripts contained in community.general were moved to the contrib-scripts GitHub repository (https://github.com/ansible-collections/community.general/pull/2696).
- ModuleHelper module utils - remove fallback when value could not be determined for a parameter (https://github.com/ansible-collections/community.general/pull/3461).
- Removed deprecated netapp module utils and doc fragments (https://github.com/ansible-collections/community.general/pull/3197).
- The nios, nios_next_ip, nios_next_network lookup plugins, the nios documentation fragment, and the nios_host_record, nios_ptr_record, nios_mx_record, nios_fixed_address, nios_zone, nios_member, nios_a_record, nios_aaaa_record, nios_network, nios_dns_view, nios_txt_record, nios_naptr_record, nios_srv_record, nios_cname_record, nios_nsgroup, and nios_network_view module have been removed from community.general 4.0.0 and were replaced by redirects to the infoblox.nios_modules collection. Please install the `infoblox.nios_modules` collection to continue using these plugins and modules, and update your FQCNs (https://github.com/ansible-collections/community.general/pull/3592).
- The vendored copy of `ipaddress` has been removed. Please use `ipaddress` from the Python 3 standard library, or from pypi. (https://github.com/ansible-collections/community.general/pull/2441).
- cpanm - removed the deprecated `system_lib` option. Use Ansible's privilege escalation mechanism instead; the option basically used `sudo` (https://github.com/ansible-collections/community.general/pull/3461).
- grove - removed the deprecated alias `message` of the `message_content` option (https://github.com/ansible-collections/community.general/pull/3461).
- proxmox - default value of `proxmox_default_behavior` changed to `no_defaults` (https://github.com/ansible-collections/community.general/pull/3461).
- proxmox_kvm - default value of `proxmox_default_behavior` changed to `no_defaults` (https://github.com/ansible-collections/community.general/pull/3461).
- runit - removed the deprecated `dist` option which was not used by the module (https://github.com/ansible-collections/community.general/pull/3461).
- telegram - removed the deprecated `msg`, `msg_format` and `chat_id` options (https://github.com/ansible-collections/community.general/pull/3461).
- xfconf - the default value of `disable_facts` changed to `true`, and the value `false` is no longer allowed. Register the module results instead (https://github.com/ansible-collections/community.general/pull/3461).

community.hashi_vault

- drop support for Python 2 and Python 3.5 (https://github.com/ansible-collections/community.hashi_vault/issues/81).
- support for the following deprecated environment variables has been removed: `VAULT_AUTH_METHOD`, `VAULT_TOKEN_PATH`, `VAULT_TOKEN_FILE`, `VAULT_ROLE_ID`, `VAULT_SECRET_ID` (https://github.com/ansible-collections/community.hashi_vault/pull/173).

Deprecated Features

Ansible-core

- ansible-test - The `--docker-no-pull` option is deprecated and has no effect.
- ansible-test - The `--no-pip-check` option is deprecated and has no effect.
- include action is deprecated in favor of include_tasks, import_tasks and import_playbook.
- module_utils' FileLock is scheduled to be removed, it is not used due to its unreliable nature.

amazon.aws

- ec2 - the boto based `ec2` module has been deprecated in favour of the boto3 based `ec2_instance` module. The `ec2` module will be removed in release 4.0.0 (https://github.com/ansible-collections/amazon.aws/pull/424).
- ec2_classic_lb - setting of the `ec2_elb` fact has been deprecated and will be removed in release 4.0.0 of the collection. The module now returns `elb` which can be accessed using the register keyword (https://github.com/ansible-collections/amazon.aws/pull/552).
- ec2_vpc_dhcp_option - The `new_config` return key has been deprecated and will be removed in a future release. It will be replaced by `dhcp_config`. Both values are returned in the interim. (https://github.com/ansible-collections/amazon.aws/pull/252)

ansible.netcommon

- network_cli - The paramiko_ssh setting `look_for_keys` was set automatically based on the values of the `password` and `private_key_file` options passed to network_cli. This option can now be set explicitly, and the automatic setting of `look_for_keys` will be removed after 2024-01-01 (https://github.com/ansible-collections/ansible.netcommon/pull/271).

ansible.windows

- win_reboot - Unreachable hosts can be ignored with `ignore_errors: True`, this ability will be removed in a future version. Use `ignore_unreachable: True` to ignore unreachable hosts instead. - https://github.com/ansible-collections/ansible.windows/issues/62
- win_updates - Deprecated the `filtered_reason` return value for each filtered up in favour of `filtered_reasons`. This has been done to show all the reasons why an update was filtered and not just the first reason.
- win_updates - Deprecated the `use_scheduled_task` option as it is no longer used.

- win_updates - Deprecated the `whitelist` and `blacklist` options in favour of `accept_list` and `reject_list` respectively to conform to the new standards used in Ansible for these types of options.

arista.eos

- Remove testing with provider for ansible-test integration jobs. This helps prepare us to move to network-ee integration tests.

cisco.ios

- Deprecated ios_bgp in favor of ios_bgp_global and ios_bgp_address_family.
- Deprecated ios_ntp modules.
- Remove testing with provider for ansible-test integration jobs. This helps prepare us to move to network-ee integration tests.

cisco.iosxr

- The iosxr_logging module has been deprecated in favor of the new iosxr_logging_global resource module and will be removed in a release after '2023-08-01'.

cisco.nxos

- Deprecated *nxos_ntp*, *nxos_ntp_options*, *nxos_ntp_auth* modules.
- The nxos_logging module has been deprecated in favor of the new nxos_logging_global resource module and will be removed in a release after '2023-08-01'.

community.aws

- dynamodb_table - DynamoDB does not support specifying non-key-attributes when creating an `ALL` index. Passing `includes` for such indexes is currently ignored but will result in failures after version 3.0.0 (https://github.com/ansible-collections/community.aws/pull/726).
- dynamodb_table - DynamoDB does not support updating the primary indexes on a table. Attempts to make such changes are currently ignored but will result in failures after version 3.0.0 (https://github.com/ansible-collections/community.aws/pull/726).
- ec2_elb - the `ec2_elb` module has been removed and redirected to the `elb_instance` module which functions identically. The original `ec2_elb` name is now deprecated and will be removed in release 3.0.0 (https://github.com/ansible-collections/community.aws/pull/586).
- ec2_elb_info - the boto based `ec2_elb_info` module has been deprecated in favour of the boto3 based `elb_classic_lb_info` module. The `ec2_elb_info` module will be removed in release 3.0.0 (https://github.com/ansible-collections/community.aws/pull/586).
- elb_classic_lb - the `elb_classic_lb` module has been removed and redirected to the `amazon.aws.ec2_elb_lb` module which functions identically.
- elb_instance - setting of the `ec2_elb` fact has been deprecated and will be removed in release 4.0.0 of the collection. See the module documentation for an alternative example using the register keyword (https://github.com/ansible-collections/community.aws/pull/773).

- iam - the boto based `iam` module has been deprecated in favour of the boto3 based `iam_user`, `iam_group` and `iam_role` modules. The `iam` module will be removed in release 3.0.0 (https://github.com/ansible-collections/community.aws/pull/664).
- iam_cert - the iam_cert module has been renamed to iam_server_certificate for consistency with the companion iam_server_certificate_info module. The usage of the module has not changed. The iam_cert alias will be removed in version 4.0.0 (https://github.com/ansible-collections/community.aws/pull/728).
- iam_server_certificate - Passing file names to the `cert`, `chain_cert` and `key` parameters has been deprecated. We recommend using a lookup plugin to read the files instead, see the documentation for an example (https://github.com/ansible-collections/community.aws/pull/735).
- iam_server_certificate - the default value for the `dup_ok` parameter is currently `false`, in version 4.0.0 this will be updated to `true`. To preserve the current behaviour explicitly set the `dup_ok` parameter to `false` (https://github.com/ansible-collections/community.aws/pull/737).
- rds - the boto based `rds` module has been deprecated in favour of the boto3 based `rds_instance` module. The `rds` module will be removed in release 3.0.0 (https://github.com/ansible-collections/community.aws/pull/663).
- rds_snapshot - the rds_snapshot module has been renamed to rds_instance_snapshot. The usage of the module has not changed. The rds_snapshot alias will be removed in version 4.0.0 (https://github.com/ansible-collections/community.aws/pull/783).
- script_inventory_ec2 - The ec2.py inventory script is being moved to a new repository. The script can now be downloaded from https://github.com/ansible-community/contrib-scripts/blob/main/inventory/ec2.py and will be removed from this collection in the 3.0 release. We recommend migrating from the script to the *amazon.aws.ec2* inventory plugin.

community.azure

- All community.azure.azure_rm_<resource>_facts modules are deprecated. Use azure.azcollection.azure_rm_<resource>_info modules instead (https://github.com/ansible-collections/community.azure/pull/24).
- All community.azure.azure_rm_<resource>_info modules are deprecated. Use azure.azcollection.azure_rm_<resource>_info modules instead (https://github.com/ansible-collections/community.azure/pull/24).
- community.azure.azure_rm_managed_disk and community.azure.azure_rm_manageddisk are deprecated. Use azure.azcollection.azure_rm_manageddisk instead (https://github.com/ansible-collections/community.azure/pull/24).
- community.azure.azure_rm_virtualmachine_extension and community.azure.azure_rm_virtualmachineextension are deprecated. Use azure.azcollection.azure_rm_virtualmachineextension instead (https://github.com/ansible-collections/community.azure/pull/24).
- community.azure.azure_rm_virtualmachine_scaleset and community.azure.azure_rm_virtualmachinescaleset are deprecated. Use azure.azcollection.azure_rm_virtualmachinescaleset instead (https://github.com/ansible-collections/community.azure/pull/24).

community.crypto

- acme_* modules - ACME version 1 is now deprecated and support for it will be removed in community.crypto 2.0.0 (https://github.com/ansible-collections/community.crypto/pull/288).

community.dns

- The hosttech_dns_records module has been renamed to hosttech_dns_record_sets. The old name will stop working in community.dns 3.0.0 (https://github.com/ansible-collections/community.dns/pull/31).

community.docker

- docker_* modules and plugins, except `docker_swarm` connection plugin and `docker_compose` and `docker_stack*`` modules - the current default ``localhost` for `tls_hostname` is deprecated. In community.docker 2.0.0 it will be computed from `docker_host` instead (https://github.com/ansible-collections/community.docker/pull/134).
- docker_container - the new `command_handling`'s default value, `compatibility`, is deprecated and will change to `correct` in community.docker 3.0.0. A deprecation warning is emitted by the module in cases where the behavior will change. Please note that ansible-core will output a deprecation warning only once, so if it is shown for an earlier task, there could be more tasks with this warning where it is not shown (https://github.com/ansible-collections/community.docker/pull/186).
- docker_container - using the special value `all` in `published_ports` has been deprecated. Use `publish_all_ports=true` instead (https://github.com/ansible-collections/community.docker/pull/210).

community.general

- Support for Ansible 2.9 and ansible-base 2.10 is deprecated, and will be removed in the next major release (community.general 5.0.0) next spring. While most content will probably still work with ansible-base 2.10, we will remove symbolic links for modules and action plugins, which will make it impossible to use them with Ansible 2.9 anymore. Please use community.general 4.x.y with Ansible 2.9 and ansible-base 2.10, as these releases will continue to support Ansible 2.9 and ansible-base 2.10 even after they are End of Life (https://github.com/ansible-community/community-topics/issues/50, https://github.com/ansible-collections/community.general/pull/3723).
- ali_instance_info - marked removal version of deprecated parameters `availability_zone` and `instance_names` (https://github.com/ansible-collections/community.general/issues/2429).
- bitbucket_* modules - `username` options have been deprecated in favor of `workspace` and will be removed in community.general 6.0.0 (https://github.com/ansible-collections/community.general/pull/2045).
- dnsimple - python-dnsimple < 2.0.0 is deprecated and support for it will be removed in community.general 5.0.0 (https://github.com/ansible-collections/community.general/pull/2946#discussion_r667624693).
- gitlab_group_members - setting `gitlab_group` to `name` or `path` is deprecated. Use `full_path` instead (https://github.com/ansible-collections/community.general/pull/3451).
- keycloak_authentication - the return value `flow` is now deprecated and will be removed in community.general 6.0.0; use `end_state` instead (https://github.com/ansible-collections/community.general/pull/3280).
- keycloak_group - the return value `group` is now deprecated and will be removed in community.general 6.0.0; use `end_state` instead (https://github.com/ansible-collections/community.general/pull/3280).

- linode - parameter `backupsenabled` is deprecated and will be removed in community.general 5.0.0 (https://github.com/ansible-collections/community.general/pull/2410).
- lxd_container - the current default value `true` of `ignore_volatile_options` is deprecated and will change to `false` in community.general 6.0.0 (https://github.com/ansible-collections/community.general/pull/3429).
- serverless - deprecating parameter `functions` because it was not used in the code (https://github.com/ansible-collections/community.general/pull/2845).
- xfconf - deprecate the `get` state. The new module `xfconf_info` should be used instead (https://github.com/ansible-collections/community.general/pull/3049).

community.grafana

- grafana_dashboard lookup - Providing a mangled version of the API key is no longer preferred.

community.hashi_vault

- hashi_vault collection - support for Python 2 will be dropped in version `2.0.0` of `community.hashi_vault` (https://github.com/ansible-collections/community.hashi_vault/issues/81).
- hashi_vault collection - support for Python 3.5 will be dropped in version `2.0.0` of `community.hashi_vault` (https://github.com/ansible-collections/community.hashi_vault/issues/81).
- lookup hashi_vault - the `[lookup_hashi_vault]` section in the `ansible.cfg` file is deprecated and will be removed in collection version `3.0.0`. Instead, the section `[hashi_vault_collection]` can be used, which will apply to all plugins in the collection going forward (https://github.com/ansible-collections/community.hashi_vault/pull/144).

community.kubernetes

- The `community.kubernetes` collection is being renamed to `kubernetes.core`. All content in the collection has been replaced by deprecated redirects to `kubernetes.core`. If you are using FQCNs starting with `community.kubernetes`, please update them to `kubernetes.core` (https://github.com/ansible-collections/community.kubernetes/pull/439).

community.vmware

- vmware_guest_vnc - Sphere 7.0 removed the built-in VNC server (https://docs.vmware.com/en/VMware-vSphere/7.0/rn/vsphere-esxi-vcenter-server-70-release-notes.html#productsupport).

inspur.sm

- add_ad_group - This feature will be removed in inspur.sm.add_ad_group 3.0.0. replaced with inspur.sm.ad_group.
- add_ldap_group - This feature will be removed in inspur.sm.add_ldap_group 3.0.0. replaced with inspur.sm.ldap_group.
- add_user - This feature will be removed in inspur.sm.add_user 3.0.0. replaced with inspur.sm.user.
- add_user_group - This feature will be removed in inspur.sm.add_user_group 3.0.0. replaced with inspur.sm.user_group.

- del_ad_group - This feature will be removed in inspur.sm.del_ad_group 3.0.0. replaced with inspur.sm.ad_group.
- del_ldap_group - This feature will be removed in inspur.sm.del_ldap_group 3.0.0. replaced with inspur.sm.ldap_group.
- del_user - This feature will be removed in inspur.sm.del_user 3.0.0. replaced with inspur.sm.user.
- del_user_group - This feature will be removed in inspur.sm.del_user_group 3.0.0. replaced with inspur.sm.user_group.
- edit_ad_group - This feature will be removed in inspur.sm.edit_ad_group 3.0.0. replaced with inspur.sm.ad_group.
- edit_ldap_group - This feature will be removed in inspur.sm.edit_ldap_group 3.0.0. replaced with inspur.sm.ldap_group.
- edit_user - This feature will be removed in inspur.sm.edit_user 3.0.0. replaced with inspur.sm.user.
- edit_user_group - This feature will be removed in inspur.sm.edit_user_group 3.0.0. replaced with inspur.sm.user_group.

junipernetworks.junos

- Deprecated router_id from ospfv2 resource module.
- Deprecated router_id from ospfv3 resource module.
- The junos_logging module has been deprecated in favor of the new junos_logging_global resource module and will be removed in a release after '2023-08-01'.

vyos.vyos

- The vyos_logging module has been deprecated in favor of the new vyos_logging_global resource module and will be removed in a release after "2023-08-01".

1.3.5 Ansible 4 Porting Guide

We suggest you read this page along with the Ansible 4 Changelog to understand what updates you may need to make.

Playbook

- The `jinja2_native` setting now does not affect the template module which implicitly returns strings. For the template lookup there is a new argument `jinja2_native` (off by default) to control that functionality. The rest of the Jinja2 expressions still operate based on the `jinja2_native` setting.

Command Line

- The `ansible-galaxy login` command has been removed, as the underlying API it used for GitHub auth has been shut down. Publishing roles or collections to Galaxy with `ansible-galaxy` now requires that a Galaxy API token be passed to the CLI using a token file (default location `~/.ansible/galaxy_token`) or (insecurely) with the `--token` argument to `ansible-galaxy`.

Deprecated

The constant `ansible.module_utils.basic._CHECK_ARGUMENT_TYPES_DISPATCHER` is deprecated. Use *`ansible.module_utils.common.parameters.DEFAULT_TYPE_VALIDATORS`* instead.

Breaking Changes

Changes to `AnsibleModule`

With the move to *`ArgumentSpecValidator`* for performing argument spec validation, the following private methods in `AnsibleModule` have been removed:

- `_check_argument_types()`
- `_check_argument_values()`
- `_check_arguments()`
- `_check_mutually_exclusive()` –> *`ansible.module_utils.common.validation.check_mutually_exclusive()`*
- `_check_required_arguments()` –> *`ansible.module_utils.common.validation.check_required_arguments()`*
- `_check_required_by()` –> *`ansible.module_utils.common.validation.check_required_by()`*
- `_check_required_if()` –> *`ansible.module_utils.common.validation.check_required_if()`*
- `_check_required_one_of()` –> *`ansible.module_utils.common.validation.check_required_one_of()`*

- `_check_required_together()` -> *`ansible.module_utils.common.validation.check_required_together()`*
- `_check_type_bits()` -> *`ansible.module_utils.common.validation.check_type_bits()`*
- `_check_type_bool()` -> *`ansible.module_utils.common.validation.check_type_bool()`*
- `_check_type_bytes()` -> *`ansible.module_utils.common.validation.check_type_bytes()`*
- `_check_type_dict()` -> *`ansible.module_utils.common.validation.check_type_dict()`*
- `_check_type_float()` -> *`ansible.module_utils.common.validation.check_type_float()`*
- `_check_type_int()` -> *`ansible.module_utils.common.validation.check_type_int()`*
- `_check_type_jsonarg()` -> *`ansible.module_utils.common.validation.check_type_jsonarg()`*
- `_check_type_list()` -> *`ansible.module_utils.common.validation.check_type_list()`*
- `_check_type_path()` -> *`ansible.module_utils.common.validation.check_type_path()`*
- `_check_type_raw()` -> *`ansible.module_utils.common.validation.check_type_raw()`*
- `_check_type_str()` -> *`ansible.module_utils.common.validation.check_type_str()`*
- `_count_terms()` -> *`ansible.module_utils.common.validation.count_terms()`*
- `_get_wanted_type()`
- `_handle_aliases()`
- `_handle_no_log_values()`
- `_handle_options()`
- `_set_defaults()`
- `_set_fallbacks()`

Modules or plugins using these private methods should use the public functions in *`ansible.module_utils.common.validation`* or *`ArgumentSpecValidator.validate()`* if no public function was listed above.

Changes to `ansible.module_utils.common.parameters`

The following functions in *`ansible.module_utils.common.parameters`* are now private and should not be used directly. Use *`ArgumentSpecValidator.validate()`* instead.

- `list_no_log_values`
- `list_deprecations`
- `handle_aliases`

Other

- **Upgrading**: If upgrading from `ansible < 2.10` or from `ansible-base` and using pip, you must `pip uninstall ansible` or `pip uninstall ansible-base` before installing `ansible-core` to avoid conflicts.
- Python 3.8 on the controller node is a soft requirement for this release. `ansible-core` 2.11 still works with the same versions of Python that `ansible-base` 2.10 worked with, however 2.11 emits a warning when running on a controller node with a Python version less than 3.8. This warning can be disabled by setting `ANSIBLE_CONTROLLER_PYTHON_WARNING=False` in your environment. `ansible-core` 2.12 will require Python 3.8 or greater.

- The configuration system now validates the `choices` field, so any settings that violate it and were ignored in 2.10 cause an error in 2.11. For example, `ANSIBLE_COLLECTIONS_ON_ANSIBLE_VERSION_MISMATCH=0` now causes an error (valid choices are `ignore`, `warn` or `error`).
- The `ansible-galaxy` command now uses `resolvelib` for resolving dependencies. In most cases this should not make a user-facing difference beyond being more performant, but we note it here for posterity and completeness.
- If you import Python `module_utils` into any modules you maintain, you may now mark the import as optional during the module payload build by wrapping the `import` statement in a `try` or `if` block. This allows modules to use `module_utils` that may not be present in all versions of Ansible or a collection, and to perform arbitrary recovery or fallback actions during module runtime.

Modules

- The `apt_key` module has explicitly defined `file` as mutually exclusive with `data`, `keyserver` and `url`. They cannot be used together anymore.
- The `meta` module now supports tags for user-defined tasks. Set the task's tags to 'always' to maintain the previous behavior. Internal `meta` tasks continue to always run.

Modules removed

The following modules no longer exist:

- No notable changes

Deprecation notices

No notable changes

Noteworthy module changes

- facts - On NetBSD, `ansible_virtualization_type` now tries to report a more accurate result than `xen` when virtualized and not running on Xen.
- facts - Virtualization facts now include `virtualization_tech_guest` and `virtualization_tech_host` keys. These are lists of virtualization technologies that a guest is a part of, or that a host provides, respectively. As an example, if you set up a host to provide both KVM and VirtualBox, both values are included in `virtualization_tech_host`. Similarly, a podman container running on a VM powered by KVM has a `virtualization_tech_guest` of `["kvm", "podman", "container"]`.
- The parameter `filter` type is changed from `string` to `list` in the setup module in order to use more than one filter. Previous behavior (using a `string`) still remains and works as a single filter.

Plugins

- inventory plugins - `CachePluginAdjudicator.flush()` now calls the underlying cache plugin's `flush()` instead of only deleting keys that it knows about. Inventory plugins should use `delete()` to remove any specific keys. As a user, this means that when an inventory plugin calls its `clear_cache()` method, facts could also be flushed from the cache. To work around this, users can configure inventory plugins to use a cache backend that is independent of the facts cache.
- callback plugins - `meta` task execution is now sent to `v2_playbook_on_task_start` like any other task. By default, only explicit meta tasks are sent there. Callback plugins can opt-in to receiving internal, implicitly created tasks to act on those as well, as noted in the plugin development documentation.
- The `choices` are now validated, so plugins that were using incorrect or incomplete choices issue an error in 2.11 if the value provided does not match. This has a simple fix: update the entries in `choices` to match reality.

Porting custom scripts

No notable changes

Porting Guide for v4.10.0

Major Changes

containers.podman

- Add podman_tag module
- Add secrets driver and driver opts support

Deprecated Features

cisco.nxos

- Deprecated nxos_snmp_community module.
- Deprecated nxos_snmp_contact module.
- Deprecated nxos_snmp_host module.
- Deprecated nxos_snmp_location module.
- Deprecated nxos_snmp_traps module.
- Deprecated nxos_snmp_user module.

junipernetworks.junos

- 'router_id' options is deprecated from junos_ospf_interfaces, junos_ospfv2 and junos_ospfv3 resource module.

Porting Guide for v4.9.0

Known Issues

purestorage.flashblade

- purefb_lag - The mac_address field in the response is not populated. This will be fixed in a future FlashBlade update.

Major Changes

fortinet.fortios

- Add real-world use cases in the example section for some configuration modules.
- Collect the current configurations of the modules and convert them into playbooks.
- Support FortiOS 7.0.1.
- Support member operation (delete/add extra members) on an object that has a list of members in it.
- Support selectors feature in `fortios_monitor_fact` and `fortios_log_fact`.

Porting Guide for v4.8.0

Breaking Changes

community.zabbix

- all roles now reference other roles and modules through their fully qualified collection names, which makes Ansible 2.10 minimum supported version for roles (see issue 477).

Deprecated Features

community.azure

- All community.azure.azure_rm_<resource>_facts modules are deprecated. Use azure.azcollection.azure_rm_<resource>_info modules instead (https://github.com/ansible-collections/community.azure/pull/24).
- All community.azure.azure_rm_<resource>_info modules are deprecated. Use azure.azcollection.azure_rm_<resource>_info modules instead (https://github.com/ansible-collections/community.azure/pull/24).
- community.azure.azure_rm_managed_disk and community.azure.azure_rm_manageddisk are deprecated. Use azure.azcollection.azure_rm_manageddisk instead (https://github.com/ansible-collections/community.azure/pull/24).

- community.azure.azure_rm_virtualmachine_extension and community.azure.azure_rm_virtualmachineextension are deprecated. Use azure.azcollection.azure_rm_virtualmachineextension instead (https://github.com/ansible-collections/community.azure/pull/24).
- community.azure.azure_rm_virtualmachine_scaleset and community.azure.azure_rm_virtualmachinescaleset are deprecated. Use azure.azcollection.azure_rm_virtualmachinescaleset instead (https://github.com/ansible-collections/community.azure/pull/24).

community.hashi_vault

- lookup hashi_vault - the `[lookup_hashi_vault]` section in the `ansible.cfg` file is deprecated and will be removed in collection version `3.0.0`. Instead, the section `[hashi_vault_collection]` can be used, which will apply to all plugins in the collection going forward (https://github.com/ansible-collections/community.hashi_vault/pull/144).

Porting Guide for v4.7.0

Major Changes

openvswitch.openvswitch

- By mistake we tagged the repo to 2.0.0 and as it wasn't intended and cannot be reverted we're releasing 2.0.1 to make the community aware of the major version update.

Deprecated Features

cisco.ios

- Deprecated ios_ntp modules.

cisco.nxos

- Deprecated *nxos_ntp*, *nxos_ntp_options*, *nxos_ntp_auth* modules.

community.vmware

- vmware_guest_vnc - Sphere 7.0 removed the built-in VNC server (https://docs.vmware.com/en/VMware-vSphere/7.0/rn/vsphere-esxi-vcenter-server-70-release-notes.html#productsupport).

junipernetworks.junos

- Deprecated router_id from ospfv2 resource module.

Porting Guide for v4.6.0

Major Changes

containers.podman

- Add systemd generation for pods
- Generate systemd service files for containers

gluster.gluster

- enable client.ssl,server.ssl before starting the gluster volume (https://github.com/gluster/gluster-ansible-collection/pull/19)

Deprecated Features

community.grafana

- grafana_dashboard lookup - Providing a mangled version of the API key is no longer preferred.

Porting Guide for v4.5.0

Major Changes

hetzner.hcloud

- Introduction of placement groups

ovirt.ovirt

- remove_stale_lun - Add role for removing stale LUN (https://bugzilla.redhat.com/1966873).

Deprecated Features

ansible.netcommon

- network_cli - The paramiko_ssh setting `look_for_keys` was set automatically based on the values of the `password` and `private_key_file` options passed to network_cli. This option can now be set explicitly, and the automatic setting of `look_for_keys` will be removed after 2024-01-01 (https://github.com/ansible-collections/ansible.netcommon/pull/271).

cisco.ios

- Deprecated ios_bgp in favor of ios_bgp_global and ios_bgp_address_family.
- Remove testing with provider for ansible-test integration jobs. This helps prepare us to move to network-ee integration tests.

junipernetworks.junos

- Deprecated router_id from ospfv3 resource module.

Porting Guide for v4.4.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.

Deprecated Features

cisco.iosxr

- The iosxr_logging module has been deprecated in favor of the new iosxr_logging_global resource module and will be removed in a release after '2023-08-01'.

cisco.nxos

- The nxos_logging module has been deprecated in favor of the new nxos_logging_global resource module and will be removed in a release after '2023-08-01'.

community.docker

- docker_container - the new `command_handling`'s default value, `compatibility`, is deprecated and will change to `correct` in community.docker 3.0.0. A deprecation warning is emitted by the module in cases where the behavior will change. Please note that ansible-core will output a deprecation warning only once, so if it is shown for an earlier task, there could be more tasks with this warning where it is not shown (https://github.com/ansible-collections/community.docker/pull/186).

junipernetworks.junos

- The junos_logging module has been deprecated in favor of the new junos_logging_global resource module and will be removed in a release after '2023-08-01'.

vyos.vyos

- The vyos_logging module has been deprecated in favor of the new vyos_logging_global resource module and will be removed in a release after "2023-08-01".

Porting Guide for v4.3.0

Major Changes

netapp.cloudmanager

- Adding stage environment to all modules in cloudmanager

Deprecated Features

community.hashi_vault

- hashi_vault collection - support for Python 3.5 will be dropped in version `2.0.0` of `community.hashi_vault` (https://github.com/ansible-collections/community.hashi_vault/issues/81).

Porting Guide for v4.2.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Major Changes

community.vmware

- vmware_object_custom_attributes_info - added a new module to gather custom attributes of an object (https://github.com/ansible-collections/community.vmware/pull/851).

dellemc.openmanage

- idrac_server_config_profile - Added support for exporting and importing Server Configuration Profile through HTTP/HTTPS share.
- ome_device_group - Added support for adding devices to a group using the IP addresses of the devices and group ID.

fortinet.fortios

- New module fortios_monitor_fact.
- Support Fortios 7.0.
- Support Log APIs.

Deprecated Features

- The community.kubernetes collection is being renamed to kubernetes.core. In Ansible 5, community.kubernetes will be replaced by an empty collection which has deprecated redirects for all the current content to kubernetes.core. If you are using FQCNs starting with `community.kubernetes.`, please update them to `kubernetes.core.` now. Note that kubernetes.core has been included in Ansible since Ansible 3.0.0 (https://github.com/ansible-community/community-topics/issues/22).

ansible.windows

- win_updates - Deprecated the `filtered_reason` return value for each filtered up in favour of `filtered_reasons`. This has been done to show all the reasons why an update was filtered and not just the first reason.
- win_updates - Deprecated the `use_scheduled_task` option as it is no longer used.
- win_updates - Deprecated the `whitelist` and `blacklist` options in favour of `accept_list` and `reject_list` respectively to conform to the new standards used in Ansible for these types of options.

community.general

- ali_instance_info - marked removal version of deprecated parameters `availability_zone` and `instance_names` (https://github.com/ansible-collections/community.general/issues/2429).
- serverless - deprecating parameter `functions` because it was not used in the code (https://github.com/ansible-collections/community.general/pull/2845).

community.hashi_vault

- hashi_vault collection - support for Python 2 will be dropped in version `2.0.0` of `community.hashi_vault` (https://github.com/ansible-collections/community.hashi_vault/issues/81).

Porting Guide for v4.1.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Major Changes

cloudscale_ch.cloud

- Add custom_image module

community.postgresql

- postgresql_query - the default value of the `as_single_query` option will be changed to `yes` in community.postgresql 2.0.0 (https://github.com/ansible-collections/community.postgresql/issues/85).

dellemc.openmanage

- ome_firmware_baseline - Module supports check mode, and allows the modification and deletion of firmware baselines.
- ome_firmware_catalog - Module supports check mode, and allows the modification and deletion of firmware catalogs.

fortinet.fortios

- Improve `fortios_configuration_fact` to use multiple selectors concurrently.
- Support `check_mode` in all cofigurationAPI-based modules.
- Support filtering for fact gathering modules `fortios_configuration_fact` and `fortios_monitor_fact`.
- Support moving policy in `firewall_central_snat_map`.
- Unify schemas for monitor API.

netbox.netbox

- packages is now a required Python package and gets installed through Ansible 2.10+.

Removed Features

ansible.windows

- win_reboot - Removed `shutdown_timeout` and `shutdown_timeout_sec` which has not done anything since Ansible 2.5.

Deprecated Features

ansible.windows

- win_reboot - Unreachable hosts can be ignored with `ignore_errors:  True`, this ability will be removed in a future version. Use `ignore_unreachable:  True` to ignore unreachable hosts instead. - https://github.com/ansible-collections/ansible.windows/issues/62

community.docker

- docker_* modules and plugins, except `docker_swarm` connection plugin and `docker_compose` and `docker_stack*` modules - the current default ``localhost` for `tls_hostname` is deprecated. In community.docker 2.0.0 it will be computed from `docker_host` instead (https://github.com/ansible-collections/community.docker/pull/134).

community.general

- All inventory and vault scripts will be removed from community.general in version 4.0.0. If you are referencing them, please update your references to the new contrib-scripts GitHub repository so your workflow will not break once community.general 4.0.0 is released (https://github.com/ansible-collections/community.general/pull/2697).
- The nios, nios_next_ip, nios_next_network lookup plugins, the nios documentation fragment, and the nios_host_record, nios_ptr_record, nios_mx_record, nios_fixed_address, nios_zone, nios_member, nios_a_record, nios_aaaa_record, nios_network, nios_dns_view, nios_txt_record, nios_naptr_record, nios_srv_record, nios_cname_record, nios_nsgroup, and nios_network_view module have been deprecated and will be removed from community.general 5.0.0. Please install the infoblox.nios_modules collection instead and use its plugins and modules (https://github.com/ansible-collections/community.general/pull/2458).

- The vendored copy of `ipaddress` will be removed in community.general 4.0.0. Please switch to `ipaddress` from the Python 3 standard library, or from pypi, if your code relies on the vendored version of `ipaddress` (https://github.com/ansible-collections/community.general/pull/2459).
- linode - parameter `backupsenabled` is deprecated and will be removed in community.general 5.0.0 (https://github.com/ansible-collections/community.general/pull/2410).
- lxd inventory plugin - the plugin will require `ipaddress` installed when used with Python 2 from community.general 4.0.0 on. `ipaddress` is part of the Python 3 standard library, but can be installed for Python 2 from pypi (https://github.com/ansible-collections/community.general/pull/2459).
- scaleway_security_group_rule - the module will require `ipaddress` installed when used with Python 2 from community.general 4.0.0 on. `ipaddress` is part of the Python 3 standard library, but can be installed for Python 2 from pypi (https://github.com/ansible-collections/community.general/pull/2459).

inspur.sm

- add_ad_group - This feature will be removed in inspur.sm.add_ad_group 3.0.0. replaced with inspur.sm.ad_group.
- add_ldap_group - This feature will be removed in inspur.sm.add_ldap_group 3.0.0. replaced with inspur.sm.ldap_group.
- add_user - This feature will be removed in inspur.sm.add_user 3.0.0. replaced with inspur.sm.user.
- add_user_group - This feature will be removed in inspur.sm.add_user_group 3.0.0. replaced with inspur.sm.user_group.
- del_ad_group - This feature will be removed in inspur.sm.del_ad_group 3.0.0. replaced with inspur.sm.ad_group.
- del_ldap_group - This feature will be removed in inspur.sm.del_ldap_group 3.0.0. replaced with inspur.sm.ldap_group.
- del_user - This feature will be removed in inspur.sm.del_user 3.0.0. replaced with inspur.sm.user.
- del_user_group - This feature will be removed in inspur.sm.del_user_group 3.0.0. replaced with inspur.sm.user_group.
- edit_ad_group - This feature will be removed in inspur.sm.edit_ad_group 3.0.0. replaced with inspur.sm.ad_group.
- edit_ldap_group - This feature will be removed in inspur.sm.edit_ldap_group 3.0.0. replaced with inspur.sm.ldap_group.
- edit_user - This feature will be removed in inspur.sm.edit_user 3.0.0. replaced with inspur.sm.user.
- edit_user_group - This feature will be removed in inspur.sm.edit_user_group 3.0.0. replaced with inspur.sm.user_group.

Porting Guide for v4.0.0

Known Issues

Ansible-core

- ansible-test - The `pylint` sanity test no longer correctly detects "bad" variable names for non-constants. See issue 3701 for additional details.

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_configuration_compliance_info - Issue(195592) Module may error out with the message `unable to process the request because an error occurred`. If the issue persists, report it to the system administrator.
- ome_smart_fabric - Issue(185322) Only three design types are supported by OpenManage Enterprise Modular but the module successfully creates a fabric when the design type is not supported.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

fortinet.fortios

- Modules for monitor API are not versioned yet.

Breaking Changes

Ansible-core

- Made SCM collections be reinstalled regardless of `--force` being present.
- NetBSD virtualization facts (specifically `ansible_virtualization_type`) now returns a more accurate value by checking the value of the `machdep.hypervisor` `sysctl` key. This change is breaking because in some cases previously, we would erroneously report `xen` even when the target is not running on Xen. This prevents that behavior in most cases. (https://github.com/ansible/ansible/issues/69352)
- Replaced the in-tree dependency resolver with an external implementation that pip >= 20.3 uses now by default — `resolvelib`. (https://github.com/ansible/ansible/issues/71784)
- The `meta` module now supports tags for user-defined tasks. Internal `meta` tasks continue to always run. (https://github.com/ansible/ansible/issues/64558)
- ansible-galaxy login command has been removed (see issue 71560)

ansible.netcommon

- Removed vendored ipaddress package from collection. If you use ansible_collections.ansible.netcommon.plugins.module_utils.compat.ipaddress in your collection, you will need to change this to import ipaddress instead. If your content using ipaddress supports Python 2.7, you will additionally need to make sure that the user has the ipaddress package installed. Please refer to https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_best_practices.html#importing-and-using-shared-code to see how to safely import external packages that may be missing from the user's system A backport of ipaddress for Python 2.7 is available at https://pypi.org/project/ipaddress/

community.docker

- docker_swarm - if `join_token` is specified, a returned join token with the same value will be replaced by `VALUE_SPECIFIED_IN_NO_LOG_PARAMETER`. Make sure that you do not blindly use the join tokens from the return value of this module when the module is invoked with `join_token` specified! This breaking change appears in a minor release since it is necessary to fix a security issue (https://github.com/ansible-collections/community.docker/pull/103).

community.general

- If you use Ansible 2.9 and these plugins or modules from this collection, community.general 3.0.0 results in errors when trying to use the DellEMC content by FQCN, like `community.general.idrac_firmware`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`dellemc.openmanage.idrac_firmware` for the previous example) and to make sure that you have `dellemc.openmanage` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 4.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install the `dellemc.openmanage` collection if you are using any of these plugins or modules. While ansible-base 2.10 or newer can use the redirects that community.general 3.0.0 adds, the collection they point to (such as dellemc.openmanage) must be installed for them to work.
- gitlab_deploy_key - if for an already existing key title a different public key was given as parameter nothing happened, now this changed so that the public key is updated to the new value (https://github.com/ansible-collections/community.general/pull/1661).
- java_keystore - instead of failing, now overwrites keystore if the alias (name) is changed. This was originally the intended behavior, but did not work due to a logic error. Make sure that your playbooks and roles do not depend on the old behavior of failing instead of overwriting (https://github.com/ansible-collections/community.general/issues/1671).
- java_keystore - instead of failing, now overwrites keystore if the passphrase is changed. Make sure that your playbooks and roles do not depend on the old behavior of failing instead of overwriting (https://github.com/ansible-collections/community.general/issues/1671).
- one_image - use pyone instead of python-oca (https://github.com/ansible-collections/community.general/pull/2032).
- utm_proxy_auth_profile - the `frontend_cookie_secret` return value now contains a placeholder string instead of the module's `frontend_cookie_secret` parameter (https://github.com/ansible-collections/community.general/pull/1736).

fortinet.fortios

- Generic FortiOS Module - FOS module to issue generic request with Ansible.
- Support for FOS Monitor API - several modules are new for monitor API.
- Unified Collection - The fortios collection itself will be adapting any FOS platforms.

servicenow.servicenow

- auth field now required for anything other than Basic authentication

theforeman.foreman

- All role variables are now prefixed with `foreman_` to avoid clashes with similarly named variables from roles outside this collection.

Major Changes

Ansible-core

- A collection can be reinstalled with new version requirements without using the `--force` flag. The collection's dependencies will also be updated if necessary with the new requirements. Use `--upgrade` to force transitive dependency updates.
- AnsibleModule - use `ArgumentSpecValidator` class for validating argument spec and remove private methods related to argument spec validation. Any modules using private methods should now use the `ArgumentSpecValidator` class or the appropriate validation function.
- Declared `resolvelib >= 0.5.3, < 0.6.0` a direct dependency of ansible-core. Refs: - https://github.com/sarugaku/resolvelib - https://pypi.org/p/resolvelib - https://pradyunsg.me/blog/2020/03/27/pip-resolver-testing
- It became possible to install Ansible Collections from local folders and namespaces folder similar to SCM structure with multiple collections.
- It became possible to upgrade Ansible collections from Galaxy servers using the `--upgrade` option with `ansible-galaxy collection install`.
- Support for role argument specification validation at role execution time. When a role contains an argument spec, an implicit validation task is inserted at the start of role execution.
- add `ArgumentSpecValidator` class for validating parameters against an argument spec outside of `AnsibleModule` (https://github.com/ansible/ansible/pull/73335)
- ansible-test - Tests run with the `centos6` and `default` test containers now use a PyPI proxy container to access PyPI when Python 2.6 is used. This allows tests running under Python 2.6 to continue functioning even though PyPI is discontinuing support for non-SNI capable clients.

ansible.netcommon

- Remove deprecated connection arguments from netconf_config

arista.eos

- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules* - Please refer to ansible.netcommon changelog for more details.

cisco.asa

- Please refer to ansible.netcommon *changelog <https://github.com/ansible-collections/ansible.netcommon/blob/main/changelogs/CHANGELOG.rst#ansible-netcommon-collection-release-notes>* for more details.
- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules*.

cisco.ios

- Please refer to ansible.netcommon changelog for more details.
- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules*.

cisco.iosxr

- Please refer to ansible.netcommon changelog for more details.
- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules*.
- ipaddress is no longer in ansible.netcommon. For Python versions without ipaddress (< 3.0), the ipaddress package is now required.

cisco.nxos

- Please refer to ansible.netcommon changelog for more details.
- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules*.

community.grafana

- introduce "skip_version_check" parameter in grafana_teams and grafana_folder modules (#147)

community.mysql

- mysql_replication - add deprecation warning that the `Is_Slave` and `Is_Master` return values will be replaced with `Is_Primary` and `Is_Replica` in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/pull/147).
- mysql_replication - the choices of the `state` option containing `master` will be finally replaced with the alternative `primary` choices in `community.mysql` 3.0.0, add deprecation warnings (https://github.com/ansible-collections/community.mysql/pull/150).
- mysql_replication - the mode options values `getslave`, `startslave`, `stopslave`, `resetslave`, `` resetslaveall` and the master_use_gtid option ``slave_pos`` are deprecated (see the alternative values) and will be removed in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/pull/97).
- mysql_replication - the return value `Is_Slave` and `Is_Master` will be replaced with `Is_Replica` and `Is_Primary` in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/issues/145).
- mysql_replication - the word `SLAVE` in messages returned by the module will be changed to `REPLICA` in `community.mysql` 2.0.0 (https://github.com/ansible-collections/community.mysql/issues/98).
- mysql_replication - the word `master` in messages returned by the module will be replaced with `primary` in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/issues/145).
- mysql_replication - the word `slave` in messages returned by the module replaced with `replica` (https://github.com/ansible-collections/community.mysql/issues/98).
- mysql_user - the `REQUIRESSL` is an alias for the `ssl` key in the `tls_requires` option in `community.mysql` 2.0.0 and support will be dropped altogether in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/issues/121).

fortinet.fortios

- New module fortios_configuration_fact
- New module fortios_json_generic
- New module fortios_monitor
- New module fortios_monitor_fact

junipernetworks.junos

- Please refer to ansible.netcommon changelog for more details.
- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules*.

netapp.ontap

- na_ontap_autosupport - Added REST support to the module.

openvswitch.openvswitch

- There is no major changes for this particular release and it was tagged by mistake and cannot be reverted.

servicenow.servicenow

- refactored client to inherit from AnsibleModule
- supports OpenID Connect authentication protocol
- supports bearer tokens for authentication

vyos.vyos

- Please refer to ansible.netcommon changelog for more details.
- Requires ansible.netcommon v2.0.0+ to support *ansible_network_single_user_mode* and *ansible_network_import_modules*
- ipaddress is no longer in ansible.netcommon. For Python versions without ipaddress (< 3.0), the ipaddress package is now required.

Removed Features

Ansible-core

- Removed *SharedPluginLoaderObj* class from ansible.plugins.strategy. It was deprecated in favor of using the standard plugin loader.
- Removed *_get_item()* alias from callback plugin base class which had been deprecated in favor of *_get_item_label()*.
- The "user" parameter was previously deprecated and is now removed in favor of "scope"
- The deprecated `ansible.constants.BECOME_METHODS` has been removed.
- The deprecated `ansible.constants.get_config()` has been removed.
- The deprecated `ansible.constants.mk_boolean()` has been removed.
- *with_** loops are no longer optimized for modules whose *name* parameters can take lists (mostly package managers). Use *name* instead of looping over individual names with *with_items* and friends.

community.general

- The `ome_device_info`, `idrac_firmware` and `idrac_server_config_profile` modules have now been migrated from community.general to the dellemc.openmanage Ansible collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.idrac_firmware` → `dellemc.openmanage.idrac_firmware`) and make sure to install the dellemc.openmanage collection.

- The deprecated ali_instance_facts module has been removed. Use ali_instance_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated gluster_heal_info module has been removed. Use gluster.gluster.gluster_heal_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated gluster_peer module has been removed. Use gluster.gluster.gluster_peer instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated gluster_volume module has been removed. Use gluster.gluster.gluster_volume instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated helm module has been removed. Use community.kubernetes.helm instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated hpilo_facts module has been removed. Use hpilo_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated idrac_redfish_facts module has been removed. Use idrac_redfish_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated jenkins_job_facts module has been removed. Use jenkins_job_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ldap_attr module has been removed. Use ldap_attrs instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated memset_memstore_facts module has been removed. Use memset_memstore_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated memset_server_facts module has been removed. Use memset_server_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated na_ontap_gather_facts module has been removed. Use netapp.ontap.na_ontap_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated nginx_status_facts module has been removed. Use nginx_status_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated one_image_facts module has been removed. Use one_image_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated onepassword_facts module has been removed. Use onepassword_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_datacenter_facts module has been removed. Use oneview_datacenter_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_enclosure_facts module has been removed. Use oneview_enclosure_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_ethernet_network_facts module has been removed. Use oneview_ethernet_network_info instead (https://github.com/ansible-collections/community.general/pull/1924).

- The deprecated oneview_fc_network_facts module has been removed. Use oneview_fc_network_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_fcoe_network_facts module has been removed. Use oneview_fcoe_network_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_logical_interconnect_group_facts module has been removed. Use oneview_logical_interconnect_group_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_network_set_facts module has been removed. Use oneview_network_set_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated oneview_san_manager_facts module has been removed. Use oneview_san_manager_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated online_server_facts module has been removed. Use online_server_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated online_user_facts module has been removed. Use online_user_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt module has been removed. Use ovirt.ovirt.ovirt_vm instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_affinity_label_facts module has been removed. Use ovirt.ovirt.ovirt_affinity_label_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_api_facts module has been removed. Use ovirt.ovirt.ovirt_api_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_cluster_facts module has been removed. Use ovirt.ovirt.ovirt_cluster_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_datacenter_facts module has been removed. Use ovirt.ovirt.ovirt_datacenter_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_disk_facts module has been removed. Use ovirt.ovirt.ovirt_disk_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_event_facts module has been removed. Use ovirt.ovirt.ovirt_event_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_external_provider_facts module has been removed. Use ovirt.ovirt.ovirt_external_provider_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_group_facts module has been removed. Use ovirt.ovirt.ovirt_group_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_host_facts module has been removed. Use ovirt.ovirt.ovirt_host_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_host_storage_facts module has been removed. Use ovirt.ovirt.ovirt_host_storage_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_network_facts module has been removed. Use ovirt.ovirt.ovirt_network_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_nic_facts module has been removed. Use ovirt.ovirt.ovirt_nic_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_permission_facts module has been removed. Use ovirt.ovirt.ovirt_permission_info instead (https://github.com/ansible-collections/community.general/pull/1924).

- The deprecated ovirt_quota_facts module has been removed. Use ovirt.ovirt.ovirt_quota_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_scheduling_policy_facts module has been removed. Use ovirt.ovirt.ovirt_scheduling_policy_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_snapshot_facts module has been removed. Use ovirt.ovirt.ovirt_snapshot_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_storage_domain_facts module has been removed. Use ovirt.ovirt.ovirt_storage_domain_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_storage_template_facts module has been removed. Use ovirt.ovirt.ovirt_storage_template_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_storage_vm_facts module has been removed. Use ovirt.ovirt.ovirt_storage_vm_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_tag_facts module has been removed. Use ovirt.ovirt.ovirt_tag_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_template_facts module has been removed. Use ovirt.ovirt.ovirt_template_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_user_facts module has been removed. Use ovirt.ovirt.ovirt_user_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_vm_facts module has been removed. Use ovirt.ovirt.ovirt_vm_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated ovirt_vmpool_facts module has been removed. Use ovirt.ovirt.ovirt_vmpool_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated purefa_facts module has been removed. Use purestorage.flasharray.purefa_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated purefb_facts module has been removed. Use purestorage.flasharray.purefb_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated python_requirements_facts module has been removed. Use python_requirements_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated redfish_facts module has been removed. Use redfish_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated scaleway_image_facts module has been removed. Use scaleway_image_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated scaleway_ip_facts module has been removed. Use scaleway_ip_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated scaleway_organization_facts module has been removed. Use scaleway_organization_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated scaleway_security_group_facts module has been removed. Use scaleway_security_group_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated scaleway_server_facts module has been removed. Use scaleway_server_info instead (https://github.com/ansible-collections/community.general/pull/1924).

- The deprecated scaleway_snapshot_facts module has been removed. Use scaleway_snapshot_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated scaleway_volume_facts module has been removed. Use scaleway_volume_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated smartos_image_facts module has been removed. Use smartos_image_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated vertica_facts module has been removed. Use vertica_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The deprecated xenserver_guest_facts module has been removed. Use xenserver_guest_info instead (https://github.com/ansible-collections/community.general/pull/1924).
- The ovirt_facts docs fragment has been removed (https://github.com/ansible-collections/community.general/pull/1924).
- airbrake_deployment - removed deprecated `token` parameter. Use `project_id` and `project_key` instead (https://github.com/ansible-collections/community.general/pull/1926).
- bigpanda - the alias `message` has been removed. Use `deployment_message` instead (https://github.com/ansible-collections/community.general/pull/1926).
- cisco_spark, cisco_webex - the alias `message` has been removed. Use `msg` instead (https://github.com/ansible-collections/community.general/pull/1926).
- clc_aa_policy - the `wait` parameter has been removed. It did not have any effect (https://github.com/ansible-collections/community.general/pull/1926).
- datadog_monitor - the alias `message` has been removed. Use `notification_message` instead (https://github.com/ansible-collections/community.general/pull/1926).
- django_manage - the parameter `liveserver` has been removed (https://github.com/ansible-collections/community.general/pull/1926).
- idrac_redfish_config - the parameters `manager_attribute_name` and `manager_attribute_value` have been removed. Use `manager_attributes` instead (https://github.com/ansible-collections/community.general/pull/1926).
- iso_extract - the alias `thirsty` has been removed. Use `force` instead (https://github.com/ansible-collections/community.general/pull/1926).
- ldap_entry - the `params` parameter is now completely removed. Using it already triggered an error since community.general 0.1.2 (https://github.com/ansible-collections/community.general/pull/2257).
- pulp_repo - the `feed_client_cert` parameter no longer defaults to the value of the `client_cert` parameter (https://github.com/ansible-collections/community.general/pull/1926).
- pulp_repo - the `feed_client_key` parameter no longer defaults to the value of the `client_key` parameter (https://github.com/ansible-collections/community.general/pull/1926).
- pulp_repo - the alias `ca_cert` has been removed. Use `feed_ca_cert` instead (https://github.com/ansible-collections/community.general/pull/1926).
- rax - unused parameter `service` removed (https://github.com/ansible-collections/community.general/pull/2020).
- redfish modules - issuing a data modification command without specifying the ID of the target System, Chassis or Manager resource when there is more than one is no longer allowed. Use the `resource_id` option to specify the target ID (https://github.com/ansible-collections/community.general/pull/1926).
- redfish_config - the parameters `bios_attribute_name` and `bios_attribute_value` have been removed. Use `bios_attributes` instead (https://github.com/ansible-collections/community.general/pull/1926).

- syspatch - the `apply` parameter has been removed. This is the default mode, so simply removing it will not change the behavior (https://github.com/ansible-collections/community.general/pull/1926).
- xbps - the `force` parameter has been removed. It did not have any effect (https://github.com/ansible-collections/community.general/pull/1926).

community.network

- The deprecated `community.network.ce_sflow` parameters: `rate_limit`, `rate_limit_slot`, and `forward_enp_slot` have been removed (https://github.com/ansible-collections/community.network/pull/255).
- The deprecated `community.network.sros` netconf plugin has been removed. Use `nokia.sros.md` instead (https://github.com/ansible-collections/community.network/pull/255).

f5networks.f5_modules

- Removed TMOS v11 support for bigip_gtm_pool and bigip_gtm_wide_ip modules
- Removed quorum and monitor_type parameters in bigip_node module. See porting guides section at https://clouddocs.f5.com/products/orchestration/ansible/devel/usage/porting-guides.html
- Removed syslog_settings and pool_settings parameters in bigip_log_destination moduke. See porting guides section at https://clouddocs.f5.com/products/orchestration/ansible/devel/usage/porting-guides.html

fortinet.fortios

- Removed module fortios_facts
- Removed module fortios_registration_forticare
- Removed module fortios_registration_vdom
- Removed module fortios_system_config_backup_restore
- Removed module fortios_system_vmlicense

Deprecated Features

Ansible-core

- Starting in 2.14, shell and command modules will no longer have the option to warn and suggest modules in lieu of commands. The `warn` parameter to these modules is now deprecated and defaults to `False`. Similarly, the `COMMAND_WARNINGS` configuration option is also deprecated and defaults to `False`. These will be removed and their presence will become an error in 2.14.
- apt_key - the parameter `key` does not have any effect, has been deprecated and will be removed in ansible-core version 2.14 (https://github.com/ansible/ansible/pull/70319).
- psrp - Set the minimum version of `pypsrp` to `0.4.0`.

ansible.netcommon

- Deprecate cli_parse module and textfsm, ttp, xml, json parser plugins as they are moved to ansible.utils collection (https://github.com/ansible-collections/ansible.netcommon/pull/182 https://github.com/ansible-collections/ansible.utils/pull/28)

cisco.nxos

- Deprecated nxos_bgp_af in favour of nxos_bgp_address_family resource module.
- Deprecated nxos_bgp_neighbor_af in favour of nxos_bgp_neighbor_address_family resource module.

cloudscale_ch.cloud

- The aliases `server_uuids` and `server_uuid` of the servers parameter in the volume module will be removed in version 3.0.0.

community.aws

- ec2_eip - formally deprecate the `instance_id` alias for `device_id` (https://github.com/ansible-collections/community.aws/pull/349).
- ec2_vpc_endpoint - deprecate the policy_file option and recommend using policy with a lookup (https://github.com/ansible-collections/community.aws/pull/366).
- ec2_vpc_endpoint_info - the `query` option has been deprecated and will be removed after 2022-12-01 (https://github.com/ansible-collections/community.aws/pull/346). The ec2_vpc_endpoint_info now defaults to listing information about endpoints. The ability to search for information about available services has been moved to the dedicated module `ec2_vpc_endpoint_service_info`.

community.crypto

- acme module_utils - the `acme` module_utils (`ansible_collections.community.crypto.plugins.module_utils.acme`) is deprecated and will be removed in community.crypto 2.0.0. Use the new Python modules in the `acme` package instead (`ansible_collections.community.crypto.plugins.module_utils.acme.xxx`) (https://github.com/ansible-collections/community.crypto/pull/184).
- acme_account_info - when `retrieve_orders=url_list`, `orders` will no longer be returned in community.crypto 2.0.0. Use `order_uris` instead (https://github.com/ansible-collections/community.crypto/pull/178).

community.general

- apt_rpm - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- composer - deprecated invalid parameter aliases `working-dir`, `global-command`, `prefer-source`, `prefer-dist`, `no-dev`, `no-scripts`, `no-plugins`, `optimize-autoloader`, `classmap-authoritative`, `apcu-autoloader`, `ignore-platform-reqs`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).

- cpanm - parameter `system_lib` deprecated in favor of using `become` (https://github.com/ansible-collections/community.general/pull/2218).
- github_deploy_key - deprecated invalid parameter alias `2fa_token`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- grove - the option `message` will be removed in community.general 4.0.0. Use the new option `message_content` instead (https://github.com/ansible-collections/community.general/pull/1929).
- homebrew - deprecated invalid parameter alias `update-brew`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- homebrew_cask - deprecated invalid parameter alias `update-brew`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- opkg - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- pacman - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- puppet - deprecated undocumented parameter `show_diff`, will be removed in 7.0.0. (https://github.com/ansible-collections/community.general/pull/1927).
- runit - unused parameter `dist` marked for deprecation (https://github.com/ansible-collections/community.general/pull/1830).
- slackpkg - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- urpmi - deprecated invalid parameter aliases `update-cache` and `no-recommends`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- xbps - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- xfconf - returning output as facts is deprecated, this will be removed in community.general 4.0.0. Please register the task output in a variable and use it instead. You can already switch to the new behavior now by using the new `disable_facts` option (https://github.com/ansible-collections/community.general/pull/1747).

community.vmware

- vmware_vmkernel_ip_config - deprecate in favor of vmware_vmkernel (https://github.com/ansible-collections/community.vmware/pull/667).

f5networks.f5_modules

- Support for Python versions earlier than 3.5 is being deprecated

1.3.6 Ansible 3 Porting Guide

Ansible 3 is based on Ansible-Base 2.10, which is the same major release as Ansible 2.10. Therefore, there is no section on ansible-base in this porting guide. If you are upgrading from Ansible 2.9, please first consult the Ansible 2.10 porting guide before continuing with the Ansible 3 porting guide.

We suggest you read this page along with the Ansible 3 Changelog to understand what updates you may need to make.

Porting Guide for v3.4.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.

- ome_configuration_compliance_info - Issue(195592) Module may error out with the message `unable to process the request because an error occurred`. If the issue persists, report it to the system administrator.
- ome_smart_fabric - Issue(185322) Only three design types are supported by OpenManage Enterprise Modular but the module successfully creates a fabric when the design type is not supported.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Major Changes

Ansible-base

- ansible-test - Tests run with the `centos6` and `default` test containers now use a PyPI proxy container to access PyPI when Python 2.6 is used. This allows tests running under Python 2.6 to continue functioning even though PyPI is discontinuing support for non-SNI capable clients.

community.postgresql

- postgresql_query - the default value of the `as_single_query` option will be changed to `yes` in community.postgresql 2.0.0 (https://github.com/ansible-collections/community.postgresql/issues/85).

netapp.ontap

- na_ontap_autosupport - Added REST support to the module.

Deprecated Features

community.aws

- ec2_vpc_endpoint_info - the `query` option has been deprecated and will be removed after 2022-12-01 (https://github.com/ansible-collections/community.aws/pull/346). The ec2_vpc_endpoint_info now defaults to listing information about endpoints. The ability to search for information about available services has been moved to the dedicated module `ec2_vpc_endpoint_service_info`.

community.docker

- docker_* modules and plugins, except `docker_swarm` connection plugin and `docker_compose` and `docker_stack*` modules - the current default ``localhost` for `tls_hostname` is deprecated. In community.docker 2.0.0 it will be computed from `docker_host` instead (https://github.com/ansible-collections/community.docker/pull/134).

Porting Guide for v3.3.0

Major Changes

community.mysql

- mysql_user - the `REQUIRESSL` is an alias for the `ssl` key in the `tls_requires` option in `community.mysql` 2.0.0 and support will be dropped altogether in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/issues/121).

Deprecated Features

community.vmware

- vmware_vmkernel_ip_config - deprecate in favor of vmware_vmkernel (https://github.com/ansible-collections/community.vmware/pull/667).

f5networks.f5_modules

- Support for Python versions earlier than 3.5 is being deprecated

Porting Guide for v3.2.0

Known Issues

dellemc.openmanage

- idrac_user - Issue(192043) Module may error out with the message `unable to perform the import or export operation because there are pending attribute changes or a configuration job is in progress`. Wait for the job to complete and run the task again.
- ome_configuration_compliance_info - Issue(195592) Module may error out with the message `unable to process the request because an error occurred`. If the issue persists, report it to the system administrator.
- ome_smart_fabric - Issue(185322) Only three design types are supported by OpenManage Enterprise Modular but the module successfully creates a fabric when the design type is not supported.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Breaking Changes

community.docker

- docker_swarm - if `join_token` is specified, a returned join token with the same value will be replaced by `VALUE_SPECIFIED_IN_NO_LOG_PARAMETER`. Make sure that you do not blindly use the join tokens from the return value of this module when the module is invoked with `join_token` specified! This breaking change appears in a minor release since it is necessary to fix a security issue (https://github.com/ansible-collections/community.docker/pull/103).

Deprecated Features

community.crypto

- acme module_utils - the `acme` module_utils (`ansible_collections.community.crypto.plugins.module_utils.acme`) is deprecated and will be removed in community.crypto 2.0.0. Use the new Python modules in the `acme` package instead (`ansible_collections.community.crypto.plugins.module_utils.acme.xxx`) (https://github.com/ansible-collections/community.crypto/pull/184).

Porting Guide for v3.1.0

Known Issues

dellemc.openmanage

- ome_smart_fabric - Issue(185322) Only three design types are supported by OpenManage Enterprise Modular but the module successfully creates a fabric when the design type is not supported.
- ome_smart_fabric_uplink - Issue(186024) ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.

Major Changes

community.grafana

- introduce "skip_version_check" parameter in grafana_teams and grafana_folder modules (#147)

community.mysql

- mysql_replication - the mode options values `getslave`, `startslave`, `stopslave`, `resetslave`, ``resetslaveall` and the master_use_gtid option ``slave_pos`` are deprecated (see the alternative values) and will be removed in `community.mysql` 3.0.0 (https://github.com/ansible-collections/community.mysql/pull/97).
- mysql_replication - the word `SLAVE` in messages returned by the module will be changed to `REPLICA` in `community.mysql` 2.0.0 (https://github.com/ansible-collections/community.mysql/issues/98).

Removed Features

f5networks.f5_modules

- Removed TMOS v11 support for bigip_gtm_pool and bigip_gtm_wide_ip modules
- Removed quorum and monitor_type parameters in bigip_node module. See porting guides section at https://clouddocs.f5.com/products/orchestration/ansible/devel/usage/porting-guides.html
- Removed syslog_settings and pool_settings parameters in bigip_log_destination moduke. See porting guides section at https://clouddocs.f5.com/products/orchestration/ansible/devel/usage/porting-guides.html

Deprecated Features

cloudscale_ch.cloud

- The aliases `server_uuids` and `server_uuid` of the servers parameter in the volume module will be removed in version 3.0.0.

community.aws

- ec2_eip - formally deprecate the `instance_id` alias for `device_id` (https://github.com/ansible-collections/community.aws/pull/349).
- ec2_vpc_endpoint - deprecate the policy_file option and recommend using policy with a lookup (https://github.com/ansible-collections/community.aws/pull/366).

community.crypto

- acme_account_info - when `retrieve_orders=url_list`, `orders` will no longer be returned in community.crypto 2.0.0. Use `order_uris` instead (https://github.com/ansible-collections/community.crypto/pull/178).

community.general

- apt_rpm - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- composer - deprecated invalid parameter aliases `working-dir`, `global-command`, `prefer-source`, `prefer-dist`, `no-dev`, `no-scripts`, `no-plugins`, `optimize-autoloader`, `classmap-authoritative`, `apcu-autoloader`, `ignore-platform-reqs`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- github_deploy_key - deprecated invalid parameter alias `2fa_token`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- grove - the option `message` will be removed in community.general 4.0.0. Use the new option `message_content` instead (https://github.com/ansible-collections/community.general/pull/1929).
- homebrew - deprecated invalid parameter alias `update-brew`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).

- homebrew_cask - deprecated invalid parameter alias `update-brew`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- opkg - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- pacman - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- puppet - deprecated undocumented parameter `show_diff`, will be removed in 7.0.0. (https://github.com/ansible-collections/community.general/pull/1927).
- runit - unused parameter `dist` marked for deprecation (https://github.com/ansible-collections/community.general/pull/1830).
- slackpkg - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- urpmi - deprecated invalid parameter aliases `update-cache` and `no-recommends`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- xbps - deprecated invalid parameter alias `update-cache`, will be removed in 5.0.0 (https://github.com/ansible-collections/community.general/pull/1927).
- xfconf - returning output as facts is deprecated, this will be removed in community.general 4.0.0. Please register the task output in a variable and use it instead. You can already switch to the new behavior now by using the new `disable_facts` option (https://github.com/ansible-collections/community.general/pull/1747).

Porting Guide for v3.0.0

Known Issues

dellemc.openmanage

- Issue 1(186024): ome_smart_fabric_uplink module does not allow the creation of multiple uplinks of the same name even though this is supported by OpenManage Enterprise Modular. If an uplink is created using the same name as an existing uplink, the existing uplink is modified.
- Issue 2(187956): If an invalid job_id is provided, idrac_lifecycle_controller_job_status_info returns an error message. This error message does not contain information about the exact issue with the invalid job_id.
- Issue 3(188267): While updating the iDRAC firmware, the idrac_firmware module completes execution before the firmware update job is completed. An incorrect message is displayed in the task output as 'DRAC WSMAN endpoint returned HTTP code '400' Reason 'Bad Request''. This issue may occur if the target iDRAC firmware version is less than 3.30.30.30

Breaking Changes

Ansible-base

- ansible-galaxy login command has been removed (see issue 71560)

ansible.utils

- If added custom sub plugins in your collection move from old location *plugins/<sub-plugin-name>* to the new location *plugins/sub_plugins/<sub-plugin-name>* and update the imports as required
- Move sub plugins cli_parsers, fact_diff and validate to *plugins/sub_plugins* folder
- The *cli_parsers* sub plugins folder name is changed to *cli_parse* to have consistent naming convention, that is all the cli_parse subplugins will now be in *plugins/sub_plugins/cli_parse* folder

cloudscale_ch.cloud

- floating_ip - `name` is required for assigning a new floating IP.

community.general

- If you use Ansible 2.9 and the Google cloud plugins or modules from this collection, community.general 2.0.0 results in errors when trying to use the Google cloud content by FQCN, like `community.general.gce_img`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.google.gce_img` for the previous example) and to make sure that you have `community.google` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install the `community.google` or `google.cloud` collections if you are using any of the Google cloud plugins or modules. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (such as community.google) must be installed for them to work.
- If you use Ansible 2.9 and the Kubevirt plugins or modules from this collection, community.general 2.0.0 results in errors when trying to use the Kubevirt content by FQCN, like `community.general.kubevirt_vm`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.kubevirt.kubevirt_vm` for the previous example) and to make sure that you have `community.kubevirt` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install the `community.kubevirt` collection if you are using any of the Kubevirt plugins or modules. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (such as community.google) must be installed for them to work.
- If you use Ansible 2.9 and the `docker` plugins or modules from this collections, community.general 2.0.0 results in errors when trying to use the docker content by FQCN, like `community.general.docker_container`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.docker.docker_container` for the previous example) and to make sure that you have `community.docker` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install `community.docker` if you are using any of the `docker` plugins or modules. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (community.docker) must be installed for them to work.
- If you use Ansible 2.9 and the `hashi_vault` lookup plugin from this collections, community.general 2.0.0 results in errors when trying to use the Hashi Vault content by FQCN, like `community.general.hashi_vault`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your inventories, variable files, playbooks and roles manually to use the new FQCN (`community.hashi_vault.hashi_vault`) and to make sure that you have `community.hashi_vault` installed.

If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install `community.hashi_vault` if you are using the `hashi_vault` plugin. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (community.hashi_vault) must be installed for them to work.

- If you use Ansible 2.9 and the `hetzner` modules from this collections, community.general 2.0.0 results in errors when trying to use the hetzner content by FQCN, like `community.general.hetzner_firewall`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.hrobot.firewall` for the previous example) and to make sure that you have `community.hrobot` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install `community.hrobot` if you are using any of the `hetzner` modules. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (community.hrobot) must be installed for them to work.

- If you use Ansible 2.9 and the `oc` connection plugin from this collections, community.general 2.0.0 results in errors when trying to use the oc content by FQCN, like `community.general.oc`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your inventories, variable files, playbooks and roles manually to use the new FQCN (`community.okd.oc`) and to make sure that you have `community.okd` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install `community.okd` if you are using the `oc` plugin. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (community.okd) must be installed for them to work.

- If you use Ansible 2.9 and the `postgresql` modules from this collections, community.general 2.0.0 results in errors when trying to use the postgresql content by FQCN, like `community.general.postgresql_info`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.postgresql.postgresql_info` for the previous example) and to make sure that you have `community.postgresql` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.general manually, you need to make sure to also install `community.postgresql` if you are using any of the `postgresql` modules. While ansible-base 2.10 or newer can use the redirects that community.general 2.0.0 adds, the collection they point to (community.postgresql) must be installed for them to work.

- The Google cloud inventory script `gce.py` has been migrated to the `community.google` collection. Install the `community.google` collection in order to continue using it.
- archive - remove path folder itself when `remove` parameter is true (https://github.com/ansible-collections/community.general/issues/1041).
- log_plays callback - add missing information to the logs generated by the callback plugin. This changes the log message format (https://github.com/ansible-collections/community.general/pull/442).
- passwordstore lookup plugin - now parsing a password store entry as YAML if possible, skipping the first line (which by convention only contains the password and nothing else). If it cannot be parsed as YAML, the old `key: value` parser will be used to process the entry. Can break backwards compatibility if YAML formatted code was parsed in a non-YAML interpreted way, for example `foo: [bar, baz]`, will become a list with two elements in the new version, but a string `'[bar, baz]'` in the old (https://github.com/ansible-collections/community.general/issues/1673).
- pkgng - passing `name: *` with `state: absent` will no longer remove every installed package from the system. It is now a noop. (https://github.com/ansible-collections/community.general/pull/569).
- pkgng - passing `name: *` with `state: latest` or `state: present` will no longer install every package from the configured package repositories. Instead, `name: *, state: latest` will upgrade all already-installed packages, and `name: *, state: present` is a noop. (https://github.com/ansible-collections/community.general/pull/569).

- proxmox_kvm - recognize `force=yes` in conjunction with `state=absent` to forcibly remove a running VM (https://github.com/ansible-collections/community.general/pull/849).
- utm_proxy_auth_profile - the `frontend_cookie_secret` return value now contains a placeholder string instead of the module's `frontend_cookie_secret` parameter (https://github.com/ansible-collections/community.general/pull/1736).

community.hashi_vault

- hashi_vault - the `VAULT_ADDR` environment variable is now checked last for the `url` parameter. For details on which use cases are impacted, see (https://github.com/ansible-collections/community.hashi_vault/issues/8).

community.hrobot

- firewall - now requires the ipaddress library (https://github.com/ansible-collections/community.hrobot/pull/2).

community.network

- If you use Ansible 2.9 and the FortiOS modules from this collection, community.network 2.0.0 results in errors when trying to use the FortiOS content by FQCN, like `community.network.fmgr_device`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.fortios.fmgr_device` for the previous example) and to make sure that you have `community.fortios` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.network manually, you need to make sure to also install `community.fortios` if you are using any of the FortiOS modules. While ansible-base 2.10 or newer can use the redirects that community.network 2.0.0 adds, the collection they point to (community.fortios) must be installed for them to work.
- If you use Ansible 2.9 and the `cp_publish` module from this collection, community.network 2.0.0 results in errors when trying to use the module by FQCN, i.e. `community.network.cp_publish`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`check_point.mgmt.cp_mgmt_publish`) and to make sure that you have `check_point.mgmt` installed. If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.network manually, you need to make sure to also install `check_point.mgmt` if you are using the `cp_publish` module. While ansible-base 2.10 or newer can use the redirects that community.network 2.0.0 adds, the collection they point to (check_point.mgmt) must be installed for them to work.
- If you use Ansible 2.9 and the `fortimanager` httpapi plugin from this collection, community.network 2.0.0 results in errors when trying to use it by FQCN (`community.network.fortimanager`). Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCN `fortinet.fortimanager.fortimanager` and to make sure that you have `fortinet.fortimanager` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.network manually, you need to make sure to also install `fortinet.fortimanager` if you are using the `fortimanager` httpapi plugin. While ansible-base 2.10 or newer can use the redirect that community.network 2.0.0 adds, the collection they point to (fortinet.fortimanager) must be installed for it to work.
- If you use Ansible 2.9 and the `nso` modules from this collection, community.network 2.0.0 results in errors when trying to use the nso content by FQCN, like `community.network.nso_config`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`cisco.nso.nso_config` for the previous example) and to make sure that you have `cisco.nso` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.network manually, you need to make sure to also install `cisco.nso` if you are using any of the `nso` modules. While ansible-base 2.10 or newer can use the redirects that community.network 2.0.0 adds, the collection they point to (cisco.nso) must be installed for them to work.

- If you use Ansible 2.9 and the `routeros` plugins or modules from this collections, community.network 2.0.0 results in errors when trying to use the routeros content by FQCN, like `community.network.routeros_command`. Since Ansible 2.9 is not able to use redirections, you will have to adjust your playbooks and roles manually to use the new FQCNs (`community.routeros.command` for the previous example) and to make sure that you have `community.routeros` installed.

  If you use ansible-base 2.10 or newer and did not install Ansible 3.0.0, but installed (and/or upgraded) community.network manually, you need to make sure to also install `community.routeros` if you are using any of the `routeros` plugins or modules. While ansible-base 2.10 or newer can use the redirects that community.network 2.0.0 adds, the collection they point to (community.routeros) must be installed for them to work.

- cnos_static_route - move ipaddress import from ansible.netcommon to builtin or package before ipaddress is removed from ansible.netcommon. You need to make sure to have the ipaddress package installed if you are using this module on Python 2.7 (https://github.com/ansible-collections/community.network/pull/129).

dellemc.os10

- os10_bgp - Changed "subnet" key as list format instead of dictionary format under "listen" key to support multiple neighbor prefix for listen command
- os10_bgp - Changed "vrf" key as list format instead of dictionary format to support multiple VRF in router BGP and changed the "vrf" key name to "vrfs"

ngine_io.cloudstack

- Authentication option using INI files for example `cloudstack.ini`, has been removed. The only supported option to authenticate is by using the module params with fallback to the ENV variables.
- default zone deprecation - The *zone* param default value, across multiple modules, has been deprecated due to unreliable API (https://github.com/ngine-io/ansible-collection-cloudstack/pull/62).

Major Changes

cisco.aci

- Change certificate_name to name in aci_aaa_user_certificate module for query operation

community.general

- For community.general 3.0.0, the `ome_device_info`, `idrac_firmware` and `idrac_server_config_profile` modules will be moved to the dellemc.openmanage collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use the DellEMC modules mentioned above from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `dellemc.openmanage.` instead of `community.general.`, for example replace `community.general.ome_device_info` in a task by `dellemc.openmanage.ome_device_info`.

If you use ansible-base and installed `community.general` manually and rely on the DellEMC modules mentioned above, you have to make sure to install the `dellemc.openmanage` collection as well. If you are using FQCNs, for example `community.general.ome_device_info` instead of `ome_device_info`, it will continue working, but we still recommend to adjust the FQCNs as well.

- The community.general collection no longer depends on the ansible.netcommon collection (https://github.com/ansible-collections/community.general/pull/1561).
- The community.general collection no longer depends on the ansible.posix collection (https://github.com/ansible-collections/community.general/pull/1157).

community.kubernetes

- k8s - Add support for template parameter (https://github.com/ansible-collections/community.kubernetes/pull/230).
- k8s_* - Add support for vaulted kubeconfig and src (https://github.com/ansible-collections/community.kubernetes/pull/193).

community.okd

- Add custom k8s module, integrate better Molecule tests (https://github.com/ansible-collections/community.okd/pull/7).
- Add downstream build scripts to build redhat.openshift (https://github.com/ansible-collections/community.okd/pull/20).
- Add openshift connection plugin, update inventory plugin to use it (https://github.com/ansible-collections/community.okd/pull/18).
- Add openshift_process module for template rendering and optional application of rendered resources (https://github.com/ansible-collections/community.okd/pull/44).
- Add openshift_route module for creating routes from services (https://github.com/ansible-collections/community.okd/pull/40).
- Initial content migration from community.kubernetes (https://github.com/ansible-collections/community.okd/pull/3).
- openshift_auth - new module (migrated from k8s_auth in community.kubernetes) (https://github.com/ansible-collections/community.okd/pull/33).

dellemc.openmanage

- Removed the existing deprecated modules.
- Standardization of ten iDRAC ansible modules based on ansible guidelines.
- Support for OpenManage Enterprise Modular.

dellemc.os10

- os10_bgp - Enhanced router bgp keyword support for non-default vrf which are supported for default vrf and additional keyword to support both default and non-default vrf
- os10_snmp role - Added support for snmp V3 features in community, group, host, engineID

f5networks.f5_modules

- Add phone home Teem integration into all modules, functionality can be disabled by setting up F5_TEEM environment variable or no_f5_teem provider parameter
- Added async_timeout parameter to bigip_ucs_fetch module to allow customization of module wait for async interface
- Changed bigip_ucs_fetch module to use asynchronous interface when generating UCS files

kubernetes.core

- Add changelog and fragments and document changelog process (https://github.com/ansible-collections/kubernetes.core/pull/131).
- helm - New module for managing Helm charts (https://github.com/ansible-collections/kubernetes.core/pull/61).
- helm_info - New module for retrieving Helm chart information (https://github.com/ansible-collections/kubernetes.core/pull/61).
- helm_plugin - new module to manage Helm plugins (https://github.com/ansible-collections/kubernetes.core/pull/154).
- helm_plugin_info - new modules to gather information about Helm plugins (https://github.com/ansible-collections/kubernetes.core/pull/154).
- helm_repository - New module for managing Helm repositories (https://github.com/ansible-collections/kubernetes.core/pull/61).
- k8s - Add support for template parameter (https://github.com/ansible-collections/kubernetes.core/pull/230).
- k8s - Inventory source migrated from Ansible 2.9 to Kubernetes collection.
- k8s - Lookup plugin migrated from Ansible 2.9 to Kubernetes collection.
- k8s - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_* - Add support for vaulted kubeconfig and src (https://github.com/ansible-collections/kubernetes.core/pull/193).
- k8s_auth - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_config_resource_name - Filter plugin migrated from Ansible 2.9 to Kubernetes collection.
- k8s_exec - New module for executing commands on pods through Kubernetes API (https://github.com/ansible-collections/kubernetes.core/pull/14).
- k8s_exec - Return rc for the command executed (https://github.com/ansible-collections/kubernetes.core/pull/158).
- k8s_info - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_log - New module for retrieving pod logs (https://github.com/ansible-collections/kubernetes.core/pull/16).
- k8s_scale - Module migrated from Ansible 2.9 to Kubernetes collection.

- k8s_service - Module migrated from Ansible 2.9 to Kubernetes collection.
- kubectl - Connection plugin migrated from Ansible 2.9 to Kubernetes collection.
- openshift - Inventory source migrated from Ansible 2.9 to Kubernetes collection.

netbox.netbox

- nb_inventory - Add `dns_name` option that adds `dns_name` to the host when `True` and device has a primary IP address. (#394)
- nb_inventory - Add `status` as a `group_by` option. (398)
- nb_inventory - Move around `extracted_primary_ip` to allow for `config_context` or `custom_field` to overwrite. (#377)
- nb_inventory - Services are now a list of integers due to NetBox 2.10 changes. (#396)
- nb_lookup - Allow ID to be passed in and use `.get` instead of `.filter`. (#376)
- nb_lookup - Allow `api_endpoint` and `token` to be found through env. (#391)

ovirt.ovirt

- cluster_upgrade - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/94).
- disaster_recovery - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/134).
- engine_setup - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/69).
- hosted_engine_setup - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/106).
- image_template - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/95).
- infra - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/92).
- manageiq - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/97).
- ovirt_system_option_info - Add new module (https://github.com/oVirt/ovirt-ansible-collection/pull/206).
- repositories - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/96).
- shutdown_env - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/112).
- vm_infra - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/93).

servicenow.servicenow

- add new tests (find with no result, search many)
- add related tests
- add support for ServiceNOW table api display_value exclude_reference_link and suppress_pagination_header
- use new API for pysnow >=0.6.0

Removed Features

community.docker

- docker_container - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_container - the default of `networks_cli_compatible` changed to `true` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_container - the unused option `trust_image_content` has been removed (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - `state=build` has been removed. Use `present` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `container_limits`, `dockerfile`, `http_timeout`, `nocache`, `rm`, `path`, `buildargs`, `pull` have been removed. Use the corresponding suboptions of `build` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `force` option has been removed. Use the more specific `force_*` options instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `source` option is now mandatory (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `use_tls` option has been removed. Use `tls` and `validate_certs` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the default of the `build.pull` option changed to `false` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image_facts - this alias is on longer available, use `docker_image_info` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_network - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_network - the `ipam_options` option has been removed. Use `ipam_config` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_service - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm - `state=inspect` has been removed. Use `docker_swarm_info` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `constraints` option has been removed. Use `placement.constraints` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `limit_cpu` and `limit_memory` options has been removed. Use the corresponding suboptions in `limits` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `log_driver` and `log_driver_options` options has been removed. Use the corresponding suboptions in `logging` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `reserve_cpu` and `reserve_memory` options has been removed. Use the corresponding suboptions in `reservations` instead (https://github.com/ansible-collections/community.docker/pull/1).

- docker_swarm_service - the `restart_policy`, `restart_policy_attempts`, `restart_policy_delay` and `restart_policy_window` options has been removed. Use the corresponding suboptions in `restart_config` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `update_delay`, `update_parallelism`, `update_failure_action`, `update_monitor`, `update_max_failure_ratio` and `update_order` options has been removed. Use the corresponding suboptions in `update_config` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_volume - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_volume - the `force` option has been removed. Use `recreate` instead (https://github.com/ansible-collections/community.docker/pull/1).

community.general

- All Google cloud modules and plugins have now been migrated away from this collection. They can be found in either the community.google or google.cloud collections. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.gce_img` → `community.google.gce_img`) and make sure to install the community.google or google.cloud collections as appropriate.
- All Kubevirt modules and plugins have now been migrated from community.general to the community.kubevirt Ansible collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.kubevirt_vm` → `community.kubevirt.kubevirt_vm`) and make sure to install the community.kubevirt collection.
- All `docker` modules and plugins have been removed from this collection. They have been migrated to the community.docker collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.docker_container` → `community.docker.docker_container`) and make sure to install the community.docker collection.
- All `hetzner` modules have been removed from this collection. They have been migrated to the community.hrobot collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.hetzner_firewall` → `community.hrobot.firewall`) and make sure to install the community.hrobot collection.
- All `postgresql` modules have been removed from this collection. They have been migrated to the community.postgresql collection.

  If you use ansible-base 2.10 or newer, redirections have been provided. If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.postgresql_info` → `community.postgresql.postgresql_info`) and make sure to install the community.postgresql collection.
- The Google cloud inventory script `gce.py` has been migrated to the `community.google` collection. Install the `community.google` collection in order to continue using it.
- The `hashi_vault` lookup plugin has been removed from this collection. It has been migrated to the community.hashi_vault collection. If you use ansible-base 2.10 or newer, redirections have been provided.

If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.hashi_vault` → `community.hashi_vault.hashi_vault`) and make sure to install the community.hashi_vault collection.

- The `oc` connection plugin has been removed from this collection. It has been migrated to the community.okd collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.general.oc` → `community.okd.oc`) and make sure to install the community.okd collection.

- The deprecated `actionable` callback plugin has been removed. Use the `ansible.builtin.default` callback plugin with `display_skipped_hosts = no` and `display_ok_hosts = no` options instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `foreman` module has been removed. Use the modules from the theforeman.foreman collection instead (https://github.com/ansible-collections/community.general/pull/1347) (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `full_skip` callback plugin has been removed. Use the `ansible.builtin.default` callback plugin with `display_skipped_hosts = no` option instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `gcdns_record` module has been removed. Use `google.cloud.gcp_dns_resource_record_set` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcdns_zone` module has been removed. Use `google.cloud.gcp_dns_managed_zone` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gce` module has been removed. Use `google.cloud.gcp_compute_instance` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcp_backend_service` module has been removed. Use `google.cloud.gcp_compute_backend_service` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcp_forwarding_rule` module has been removed. Use `google.cloud.gcp_compute_forwarding_rule` or `google.cloud.gcp_compute_global_forwarding_rule` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcp_healthcheck` module has been removed. Use `google.cloud.gcp_compute_health_check`, `google.cloud.gcp_compute_http_health_check` or `google.cloud.gcp_compute_https_health_check` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcp_target_proxy` module has been removed. Use `google.cloud.gcp_compute_target_http_proxy` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcp_url_map` module has been removed. Use `google.cloud.gcp_compute_url_map` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `gcspanner` module has been removed. Use `google.cloud.gcp_spanner_database` and/or `google.cloud.gcp_spanner_instance` instead (https://github.com/ansible-collections/community.general/pull/1370).
- The deprecated `github_hooks` module has been removed. Use `community.general.github_webhook` and `community.general.github_webhook_info` instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `katello` module has been removed. Use the modules from the theforeman.foreman collection instead (https://github.com/ansible-collections/community.general/pull/1347).

- The deprecated `na_cdot_aggregate` module has been removed. Use netapp.ontap.na_ontap_aggregate instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_license` module has been removed. Use netapp.ontap.na_ontap_license instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_lun` module has been removed. Use netapp.ontap.na_ontap_lun instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_qtree` module has been removed. Use netapp.ontap.na_ontap_qtree instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_svm` module has been removed. Use netapp.ontap.na_ontap_svm instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_user_role` module has been removed. Use netapp.ontap.na_ontap_user_role instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_user` module has been removed. Use netapp.ontap.na_ontap_user instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `na_cdot_volume` module has been removed. Use netapp.ontap.na_ontap_volume instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `sf_account_manager` module has been removed. Use netapp.elementsw.na_elementsw_account instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `sf_check_connections` module has been removed. Use netapp.elementsw.na_elementsw_check_connections instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `sf_snapshot_schedule_manager` module has been removed. Use netapp.elementsw.na_elementsw_snapshot_schedule instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `sf_volume_access_group_manager` module has been removed. Use netapp.elementsw.na_elementsw_access_group instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `sf_volume_manager` module has been removed. Use netapp.elementsw.na_elementsw_volume instead (https://github.com/ansible-collections/community.general/pull/1347).
- The deprecated `stderr` callback plugin has been removed. Use the `ansible.builtin.default` callback plugin with `display_failed_stderr = yes` option instead (https://github.com/ansible-collections/community.general/pull/1347).
- The redirect of the `conjur_variable` lookup plugin to `cyberark.conjur.conjur_variable` collection was removed (https://github.com/ansible-collections/community.general/pull/1346).
- The redirect of the `firewalld` module and the `firewalld` module_utils to the `ansible.posix` collection was removed (https://github.com/ansible-collections/community.general/pull/1346).
- The redirect to the `community.digitalocean` collection was removed for: the `digital_ocean` doc fragment, the `digital_ocean` module_utils, and the following modules: `digital_ocean`, `digital_ocean_account_facts`, `digital_ocean_account_info`, `digital_ocean_block_storage`, `digital_ocean_certificate`, `digital_ocean_certificate_facts`, `digital_ocean_certificate_info`, `digital_ocean_domain`, `digital_ocean_domain_facts`, `digital_ocean_domain_info`, `digital_ocean_droplet`, `digital_ocean_firewall_facts`, `digital_ocean_firewall_info`, `digital_ocean_floating_ip`, `digital_ocean_floating_ip_facts`, `digital_ocean_floating_ip_info`, `digital_ocean_image_facts`, `digital_ocean_image_info`, `digital_ocean_load_balancer_facts`,

digital_ocean_load_balancer_info, digital_ocean_region_facts, digital_ocean_region_info, digital_ocean_size_facts, digital_ocean_size_info, digital_ocean_snapshot_facts, digital_ocean_snapshot_info, digital_ocean_sshkey, digital_ocean_sshkey_facts, digital_ocean_sshkey_info, digital_ocean_tag, digital_ocean_tag_facts, digital_ocean_tag_info, digital_ocean_volume_facts, digital_ocean_volume_info (https://github.com/ansible-collections/community.general/pull/1346).

- The redirect to the `community.mysql` collection was removed for: the `mysql` doc fragment, the `mysql` module_utils, and the following modules: `mysql_db`, `mysql_info`, `mysql_query`, `mysql_replication`, `mysql_user`, `mysql_variables` (https://github.com/ansible-collections/community.general/pull/1346).
- The redirect to the `community.proxysql` collection was removed for: the `proxysql` doc fragment, and the following modules: `proxysql_backend_servers`, `proxysql_global_variables`, `proxysql_manage_config`, `proxysql_mysql_users`, `proxysql_query_rules`, `proxysql_replication_hostgroups`, `proxysql_scheduler` (https://github.com/ansible-collections/community.general/pull/1346).
- The redirect to the `infinidat.infinibox` collection was removed for: the `infinibox` doc fragment, the `infinibox` module_utils, and the following modules: `infini_export`, `infini_export_client`, `infini_fs`, `infini_host`, `infini_pool`, `infini_vol` (https://github.com/ansible-collections/community.general/pull/1346).
- conjur_variable lookup - has been moved to the `cyberark.conjur` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/570).
- digital_ocean_* - all DigitalOcean modules have been moved to the `community.digitalocean` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/622).
- infini_* - all infinidat modules have been moved to the `infinidat.infinibox` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/607).
- iptables_state - the `ANSIBLE_ASYNC_DIR` environment is no longer supported, use the `async_dir` shell option instead (https://github.com/ansible-collections/community.general/pull/1371).
- logicmonitor - the module has been removed in 1.0.0 since it is unmaintained and the API used by the module has been turned off in 2017 (https://github.com/ansible-collections/community.general/issues/539, https://github.com/ansible-collections/community.general/pull/541).
- logicmonitor_facts - the module has been removed in 1.0.0 since it is unmaintained and the API used by the module has been turned off in 2017 (https://github.com/ansible-collections/community.general/issues/539, https://github.com/ansible-collections/community.general/pull/541).
- memcached cache plugin - do not import `CacheModule``s directly. Use ``ansible.plugins.loader.cache_loader` instead (https://github.com/ansible-collections/community.general/pull/1371).
- mysql_* - all MySQL modules have been moved to the `community.mysql` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/633).
- proxysql_* - all ProxySQL modules have been moved to the `community.proxysql` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/624).
- redis cache plugin - do not import `CacheModule``s directly. Use ``ansible.plugins.loader.cache_loader` instead (https://github.com/ansible-collections/community.general/pull/1371).
- xml - when `content=attribute`, the `attribute` option is ignored (https://github.com/ansible-collections/community.general/pull/1371).

community.network

- All FortiOS modules and plugins have been removed from this collection. They have been migrated to the community.fortios collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.network.fmgr_device` → `community.fortios.fmgr_device`) and make sure to install the *community.fortios* collection.

- All `nso` modules have been removed from this collection. They have been migrated to the cisco.nso collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.network.nso_config` → `cisco.nso.nso_config`) and make sure to install the *cisco.nso* collection.

- All `routeros` modules and plugins have been removed from this collection. They have been migrated to the community.routeros collection. If you use ansible-base 2.10 or newer, redirections have been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.network.routeros_command` → `community.routeros.command`) and make sure to install the community.routeros collection.

- The `cp_publish` module has been removed from this collection. It was a duplicate of `check_point.mgmt.cp_mgmt_publish` in the check_point.mgmt collection. If you use ansible-base 2.10 or newer, redirections have been provided. If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.network.cp_publish` → `check_point.mgmt.cp_mgmt_publish`) and make sure to install the check_point.mgmt collection.

- The `fortimanager` httpapi plugin has been removed from this collection. It was a duplicate of the one in the fortinet.fortimanager collection. If you use ansible-base 2.10 or newer, a redirection has been provided.

  If you use Ansible 2.9 and installed this collection, you need to adjust the FQCNs (`community.network.fortimanager` → `fortinet.fortimanager.fortimanager`) and make sure to install the *fortinet.fortimanager* collection.

- The dependency on the `check_point.mgmt` collection has been removed. If you depend on that installing `community.network` also installs `check_point.mgmt`, you have to make sure to install `check_point.mgmt` explicitly.

- The deprecated Pluribus Networks modules `pn_cluster`, `pn_ospf`, `pn_ospfarea`, `pn_show`, `pn_trunk`, `pn_vlag`, `pn_vlan`, `pn_vrouter`, `pn_vrouterbgp`, `pn_vrouterif`, `pn_vrouterlbif` have been removed (https://github.com/ansible-collections/community.network/pull/176).

- The deprecated modules `panos_admin`, `panos_admpwd`, `panos_cert_gen_ssh`, `panos_check`, `panos_commit`, `panos_dag`, `panos_dag_tags`, `panos_import`, `panos_interface`, `panos_lic`, `panos_loadcfg`, `panos_match_rule`, `panos_mgtconfig`, `panos_nat_rule`, `panos_object`, `panos_op`, `panos_pg`, `panos_query_rules`, `panos_restart`, `panos_sag`, `panos_security_rule`, `panos_set` have been removed. Use modules from the paloaltonetworks.panos collection instead (https://github.com/ansible-collections/community.network/pull/176).

- The redirect to the `mellanox.onyx` collection was removed for: the `onyx` cliconf plugin, terminal plugin, module_utils, action plugin, doc fragment, and the following modules: `onyx_aaa`, `onyx_bfd`, `onyx_bgp`, `onyx_buffer_pool`, `onyx_command`, `onyx_config`, `onyx_facts`, `onyx_igmp`, `onyx_igmp_interface`, `onyx_igmp_vlan`, `onyx_interface`, `onyx_l2_interface`, `onyx_l3_interface`, `onyx_linkagg`, `onyx_lldp`, `onyx_lldp_interface`, `onyx_magp`, `onyx_mlag_ipl`, `onyx_mlag_vip`, `onyx_ntp`, `onyx_ntp_servers_peers`, `onyx_ospf`, `onyx_pfc_interface`, `onyx_protocol`, `onyx_ptp_global`, `onyx_ptp_interface`, `onyx_qos`, `onyx_snmp`, `onyx_snmp_hosts`, `onyx_snmp_users`, `onyx_syslog_files`, `onyx_syslog_remote`, `onyx_traffic_class`, `onyx_username`, `onyx_vlan`, `onyx_vxlan`, `onyx_wjh` (https://github.com/ansible-collections/community.network/pull/175).

- onyx - all onyx modules and plugins have been moved to the mellanox.onyx collection. Redirects have been added that will be removed in community.network 2.0.0 (https://github.com/ansible-collections/community.network/pull/83).

f5networks.f5_modules

- Removed arp_state parameter from the bigip_virtual_address module

Deprecated Features

cisco.nxos

- Deprecated *nxos_bgp* and *nxos_bgp_neighbor* modules in favor of *nxos_bgp_global* resource module.
- Deprecated *nxos_interface_ospf* in favor of *nxos_ospf_interfaces* Resource Module.
- Deprecated *nxos_smu* in favor of *nxos_rpm* module.
- The *nxos_ospf_vrf* module is deprecated by *nxos_ospfv2* and *nxos_ospfv3* Resource Modules.

community.aws

- ec2_vpc_igw_info - After 2022-06-22 the `convert_tags` parameter default value will change from `False` to `True` to match the collection standard behavior (https://github.com/ansible-collections/community.aws/pull/318).

community.docker

- docker_container - currently `published_ports` can contain port mappings next to the special value `all`, in which case the port mappings are ignored. This behavior is deprecated for community.docker 2.0.0, at which point it will either be forbidden, or this behavior will be properly implemented similar to how the Docker CLI tool handles this (https://github.com/ansible-collections/community.docker/issues/8, https://github.com/ansible-collections/community.docker/pull/60).

community.general

- The `gluster_heal_info`, `gluster_peer` and `gluster_volume` modules have migrated to the gluster.gluster collection. Ansible-base 2.10.1 adjusted the routing target to point to the modules in that collection, so we will remove these modules in community.general 3.0.0. If you use Ansible 2.9, or use FQCNs `community.general.gluster_*` in your playbooks and/or roles, please update them to use the modules from `gluster.gluster` instead.
- The ldap_attr module has been deprecated and will be removed in a later release; use ldap_attrs instead.
- django_manage - the parameter `liveserver` relates to a no longer maintained third-party module for django. It is now deprecated, and will be remove in community.general 3.0.0 (https://github.com/ansible-collections/community.general/pull/1154).
- proxmox - the default of the new `proxmox_default_behavior` option will change from `compatibility` to `no_defaults` in community.general 4.0.0. Set the option to an explicit value to avoid a deprecation warning (https://github.com/ansible-collections/community.general/pull/850).

- proxmox_kvm - the default of the new `proxmox_default_behavior` option will change from `compatibility` to `no_defaults` in community.general 4.0.0. Set the option to an explicit value to avoid a deprecation warning (https://github.com/ansible-collections/community.general/pull/850).
- syspatch - deprecate the redundant `apply` argument (https://github.com/ansible-collections/community.general/pull/360).
- xbps - the `force` option never had any effect. It is now deprecated, and will be removed in 3.0.0 (https://github.com/ansible-collections/community.general/pull/568).

community.hashi_vault

- hashi_vault - `VAULT_ADDR` environment variable for option `url` will have its precedence lowered in 1.0.0; use `ANSIBLE_HASHI_VAULT_ADDR` to intentionally override a config value (https://github.com/ansible-collections/community.hashi_vault/issues/8).
- hashi_vault - `VAULT_AUTH_METHOD` environment variable for option `auth_method` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_AUTH_METHOD` instead (https://github.com/ansible-collections/community.hashi_vault/issues/17).
- hashi_vault - `VAULT_ROLE_ID` environment variable for option `role_id` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_ROLE_ID` instead (https://github.com/ansible-collections/community.hashi_vault/issues/20).
- hashi_vault - `VAULT_SECRET_ID` environment variable for option `secret_id` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_SECRET_ID` instead (https://github.com/ansible-collections/community.hashi_vault/issues/20).
- hashi_vault - `VAULT_TOKEN_FILE` environment variable for option `token_file` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_TOKEN_FILE` instead (https://github.com/ansible-collections/community.hashi_vault/issues/15).
- hashi_vault - `VAULT_TOKEN_PATH` environment variable for option `token_path` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_TOKEN_PATH` instead (https://github.com/ansible-collections/community.hashi_vault/issues/15).

community.network

- Deprecate connection=local support for network platforms using persistent framework (https://github.com/ansible-collections/community.network/pull/120).

community.vmware

- vmware_host_firewall_manager - the creation of new rule with no `allowed_ip` entry in the `allowed_hosts` dictionary won't be allowed after 2.0.0 release.

dellemc.openmanage

- The `dellemc_get_firmware_inventory` module is deprecated and replaced with `idrac_firmware_info`.
- The `dellemc_get_system_inventory` module is deprecated and replaced with `idrac_system_info`.
- The dellemc_change_power_state module is deprecated and replaced with the redfish_powerstate module.
- The dellemc_configure_bios module is deprecated and replaced with the idrac_bios module.
- The dellemc_configure_idrac_network module is deprecated and replaced with the idrac_network module.
- The dellemc_configure_idrac_timezone module is deprecated and replaced with the idrac_timezone_ntp module.
- The dellemc_configure_idrac_users module is deprecated and replaced with the idrac_user module.
- The dellemc_delete_lc_job and dellemc_delete_lc_job_queue modules are deprecated and replaced with the idrac_lifecycle_controller_jobs module.
- The dellemc_export_lc_logs module is deprecated and replaced with the idrac_lifecycle_controller_logs module.
- The dellemc_get_lc_job_status module is deprecated and replaced with the idrac_lifecycle_controller_job_status_info module.
- The dellemc_get_lcstatus module is deprecated and replaced with the idrac_lifecycle_controller_status_info module.
- The dellemc_idrac_reset module is deprecated and replaced with the idrac_reset module.
- The dellemc_setup_idrac_syslog module is deprecated and replaced with the idrac_syslog module.

1.3.7 Ansible 2.10 Porting Guide

Warning: In Ansible 2.10, many plugins and modules have migrated to Collections on Ansible Galaxy. Your playbooks should continue to work without any changes. We recommend you start using the fully-qualified collection name (FQCN) in your playbooks as the explicit and authoritative indicator of which collection to use as some collections may contain duplicate module names. You can search the index of all modules to find the collection a module has been relocated to.

This section discusses the behavioral changes between Ansible 2.9 and Ansible 2.10.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with the Ansible Changelog for 2.10 to understand what updates you may need to make.

Since 2.10, Ansible consists of two parts:

- ansible-base, which includes the command line tools with a small selection of plugins and modules, and
- a set of collections.

The porting_2.10_guide_base is included in this porting guide. The complete list of porting guides can be found at *porting guides*.

Playbook

- Fixed a bug on boolean keywords that made random strings return 'False', now they should return an error if they are not a proper boolean Example: `diff: yes-` was returning `False`.
- A new fact, `ansible_processor_nproc` reflects the number of vcpus available to processes (falls back to the number of vcpus available to the scheduler).

Command Line

- The `ansible-galaxy login` command has been removed, as the underlying API it used for GitHub auth is being shut down. Publishing roles or collections to Galaxy through `ansible-galaxy` now requires that a Galaxy API token be passed to the CLI through a token file (default location `~/.ansible/galaxy_token`) or (insecurely) through the `--token` argument to `ansible-galaxy`.

Deprecated

- Windows Server 2008 and 2008 R2 will no longer be supported or tested in the next Ansible release, see *Are Server 2008, 2008 R2 and Windows 7 supported?*.

Modules

Warning: Links on this page may not point to the most recent versions of modules. We will update them when we can.

- Version 2.10.0 of ansible-base changed the default mode of file-based tasks to `0o600 & ~umask` when the user did not specify a `mode` parameter on file-based tasks. This was in response to a CVE report which we have reconsidered. As a result, the mode change has been reverted in 2.10.1, and mode will now default to `0o666 & ~umask` as in previous versions of Ansible.
- If you changed any tasks to specify less restrictive permissions while using 2.10.0, those changes will be unnecessary (but will do no harm) in 2.10.1.
- To avoid the issue raised in CVE-2020-1736, specify a `mode` parameter in all file-based tasks that accept it.
- `dnf` and `yum` - As of version 2.10.1, the `dnf` module (and `yum` action when it uses `dnf`) now correctly validates GPG signatures of packages (CVE-2020-14365). If you see an error such as `Failed to validate GPG signature for [package name]`, please ensure that you have imported the correct GPG key for the DNF repository and/or package you are using. One way to do this is with the `rpm_key` module. Although we discourage it, in some cases it may be necessary to disable the GPG check. This can be done by explicitly adding `disable_gpg_check: yes` in your `dnf` or `yum` task.

Noteworthy module changes

- Ansible modules created with `add_file_common_args=True` added a number of undocumented arguments which were mostly there to ease implementing certain action plugins. The undocumented arguments `src`, `follow`, `force`, `content`, `backup`, `remote_src`, `regexp`, `delimiter`, and `directory_mode` are now no longer added. Modules relying on these options to be added need to specify them by themselves.
- Ansible no longer looks for Python modules in the current working directory (typically the `remote_user`'s home directory) when an Ansible module is run. This is to fix becoming an unprivileged user on OpenBSD and to mitigate any attack vector if the current working directory is writable by a malicious user. Install any Python modules needed to run the Ansible modules on the managed node in a system-wide location or in another directory which is in the `remote_user`'s `$PYTHONPATH` and readable by the `become_user`.

Plugins

Lookup plugin names case-sensitivity

- Prior to Ansible `2.10` lookup plugin names passed in as an argument to the `lookup()` function were treated as case-insensitive as opposed to lookups invoked through `with_<lookup_name>`. `2.10` brings consistency to `lookup()` and `with_` to be both case-sensitive.

Noteworthy plugin changes

- Cache plugins in collections can be used to cache data from inventory plugins. Previously, cache plugins in collections could only be used for fact caching.
- Some undocumented arguments from `FILE_COMMON_ARGUMENTS` have been removed; plugins using these, in particular action plugins, need to be adjusted. The undocumented arguments which were removed are `src`, `follow`, `force`, `content`, `backup`, `remote_src`, `regexp`, `delimiter`, and `directory_mode`.

Action plugins which execute modules should use fully-qualified module names

- Action plugins that call modules should pass explicit, fully-qualified module names to `_execute_module()` whenever possible (eg, `ansible.builtin.file` rather than `file`). This ensures that the task's collection search order is not consulted to resolve the module. Otherwise, a module from a collection earlier in the search path could be used when not intended.

Porting custom scripts

No notable changes

Porting Guide for v2.10.7

Breaking Changes

community.general

- utm_proxy_auth_profile - the `frontend_cookie_secret` return value now contains a placeholder string instead of the module's `frontend_cookie_secret` parameter (https://github.com/ansible-collections/community.general/pull/1736).

Major Changes

- Restricting the version of the community.okd collection to 1.0.0. The previously included version, 1.0.1, had a dependency on kubernetes.core and thus required the installation of an additional collection that was not included in Ansible 2.10. Version 1.0.0 is essentially identical to 1.0.1, except that it uses community.kubernetes, which is included in Ansible 2.10.

ovirt.ovirt

- ovirt_system_option_info - Add new module (https://github.com/oVirt/ovirt-ansible-collection/pull/206).

servicenow.servicenow

- add new tests (find with no result, search many)
- add related tests
- add support for ServiceNOW table api display_value exclude_reference_link and suppress_pagination_header
- use new API for pysnow >=0.6.0

Deprecated Features

cisco.nxos

- Deprecated *nxos_bgp* and *nxos_bgp_neighbor* modules in favor of *nxos_bgp_global* resource module.

community.vmware

- vmware_host_firewall_manager - the creation of new rule with no `allowed_ip` entry in the `allowed_hosts` dictionary won't be allowed after 2.0.0 release.

Porting Guide for v2.10.6

Major Changes

community.general

- For community.general 2.0.0, the kubevirt modules will be moved to the community.kubevirt collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use kubevirt modules from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.kubevirt.` instead of `community.general.`, for example replace `community.general.kubevirt_vm` in a task by `community.kubevirt.kubevirt_vm`.

  If you use ansible-base and installed `community.general` manually and rely on the kubevirt modules, you have to make sure to install the `community.kubevirt` collection as well. If you are using FQCNs, for example `community.general.kubevirt_vm` instead of `kubevirt_vm`, it will continue working, but we still recommend to adjust the FQCNs as well.

community.network

- For community.network 2.0.0, the Cisco NSO modules will be moved to the cisco.nso collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use Cisco NSO modules from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `cisco.nso.` instead of `community.network.`, for example replace `community.network.nso_config` in a task by `cisco.nso.nso_config`.

  If you use ansible-base and installed `community.network` manually and rely on the Cisco NSO modules, you have to make sure to install the `cisco.nso` collection as well. If you are using FQCNs, for example `community.network.nso_config` instead of `nso_config`, it will continue working, but we still recommend to adjust the FQCNs as well.

- For community.network 2.0.0, the FortiOS modules will be moved to the community.fortios collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use FortiOS modules from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.fortios.` instead of `community.network.`, for example replace `community.network.fmgr_device` in a task by `community.fortios.fmgr_device`.

  If you use ansible-base and installed `community.network` manually and rely on the FortiOS modules, you have to make sure to install the `community.fortios` collection as well. If you are using FQCNs, for example `community.network.fmgr_device` instead of `fmgr_device`, it will continue working, but we still recommend to adjust the FQCNs as well.

f5networks.f5_modules

- Added async_timeout parameter to bigip_ucs_fetch module to allow customization of module wait for async interface
- Changed bigip_ucs_fetch module to use asynchronous interface when generating UCS files

Porting Guide for v2.10.5

Breaking Changes

community.hashi_vault

- hashi_vault - the `VAULT_ADDR` environment variable is now checked last for the `url` parameter. For details on which use cases are impacted, see (https://github.com/ansible-collections/community.hashi_vault/issues/8).

Major Changes

community.general

- For community.general 2.0.0, the Google modules will be moved to the community.google collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use Google modules from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.google.` instead of `community.general.`, for example replace `community.general.gcpubsub` in a task by `community.google.gcpubsub`.

  If you use ansible-base and installed `community.general` manually and rely on the Google modules, you have to make sure to install the `community.google` collection as well. If you are using FQCNs, for example `community.general.gcpubsub` instead of `gcpubsub`, it will continue working, but we still recommend to adjust the FQCNs as well.

- For community.general 2.0.0, the OC connection plugin will be moved to the community.okd collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use OC connection plugin from this collection, you will need to adjust your playbooks and roles to use FQCNs `community.okd.oc` instead of `community.general.oc`.

  If you use ansible-base and installed `community.general` manually and rely on the OC connection plugin, you have to make sure to install the `community.okd` collection as well. If you are using FQCNs, in other words `community.general.oc` instead of `oc`, it will continue working, but we still recommend to adjust this FQCN as well.

- For community.general 2.0.0, the hashi_vault lookup plugin will be moved to the community.hashi_vault collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use hashi_vault lookup plugin from this collection, you will need to adjust your playbooks and roles to use FQCNs `community.hashi_vault.hashi_vault` instead of `community.general.hashi_vault`.

  If you use ansible-base and installed `community.general` manually and rely on the hashi_vault lookup plugin, you have to make sure to install the `community.hashi_vault` collection as well. If you are using FQCNs, in other words `community.general.hashi_vault` instead of `hashi_vault`, it will continue working, but we still recommend to adjust this FQCN as well.

netbox.netbox

- nb_inventory - Add `dns_name` option that adds `dns_name` to the host when `True` and device has a primary IP address. (#394)
- nb_inventory - Add `status` as a `group_by` option. (398)
- nb_inventory - Move around `extracted_primary_ip` to allow for `config_context` or `custom_field` to overwrite. (#377)
- nb_inventory - Services are now a list of integers due to NetBox 2.10 changes. (#396)
- nb_lookup - Allow ID to be passed in and use `.get` instead of `.filter`. (#376)
- nb_lookup - Allow `api_endpoint` and `token` to be found through env. (#391)

Deprecated Features

community.aws

- ec2_vpc_igw_info - After 2022-06-22 the `convert_tags` parameter default value will change from `False` to `True` to match the collection standard behavior (https://github.com/ansible-collections/community.aws/pull/318).

community.docker

- docker_container - currently `published_ports` can contain port mappings next to the special value `all`, in which case the port mappings are ignored. This behavior is deprecated for community.docker 2.0.0, at which point it will either be forbidden, or this behavior will be properly implemented similar to how the Docker CLI tool handles this (https://github.com/ansible-collections/community.docker/issues/8, https://github.com/ansible-collections/community.docker/pull/60).

community.hashi_vault

- hashi_vault - `VAULT_ADDR` environment variable for option `url` will have its precedence lowered in 1.0.0; use `ANSIBLE_HASHI_VAULT_ADDR` to intentionally override a config value (https://github.com/ansible-collections/community.hashi_vault/issues/8).
- hashi_vault - `VAULT_AUTH_METHOD` environment variable for option `auth_method` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_AUTH_METHOD` instead (https://github.com/ansible-collections/community.hashi_vault/issues/17).
- hashi_vault - `VAULT_ROLE_ID` environment variable for option `role_id` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_ROLE_ID` instead (https://github.com/ansible-collections/community.hashi_vault/issues/20).
- hashi_vault - `VAULT_SECRET_ID` environment variable for option `secret_id` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_SECRET_ID` instead (https://github.com/ansible-collections/community.hashi_vault/issues/20).
- hashi_vault - `VAULT_TOKEN_FILE` environment variable for option `token_file` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_TOKEN_FILE` instead (https://github.com/ansible-collections/community.hashi_vault/issues/15).

- hashi_vault - `VAULT_TOKEN_PATH` environment variable for option `token_path` will be removed in 2.0.0, use `ANSIBLE_HASHI_VAULT_TOKEN_PATH` instead (https://github.com/ansible-collections/community.hashi_vault/issues/15).

Porting Guide for v2.10.4

Breaking Changes

community.hrobot

- firewall - now requires the ipaddress library (https://github.com/ansible-collections/community.hrobot/pull/2).

Major Changes

community.general

- For community.general 2.0.0, the Hetzner Robot modules will be moved to the community.hrobot collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use Hetzner Robot modules from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.hrobot.` instead of `community.general.hetzner_`, for example replace `community.general.hetzner_firewall_info` in a task by `community.hrobot.firewall_info`.

  If you use ansible-base and installed `community.general` manually and rely on the Hetzner Robot modules, you have to make sure to install the `community.hrobot` collection as well. If you are using FQCNs, i.e. `community.general.hetzner_failover_ip` instead of `hetzner_failover_ip`, it will continue working, but we still recommend to adjust the FQCNs as well.

- For community.general 2.0.0, the `docker` modules and plugins will be moved to the community.docker collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use `docker` content from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.docker.` instead of `community.general.`, for example replace `community.general.docker_container` in a task by `community.docker.docker_container`.

  If you use ansible-base and installed `community.general` manually and rely on the `docker` content, you have to make sure to install the `community.docker` collection as well. If you are using FQCNs, i.e. `community.general.docker_container` instead of `docker_container`, it will continue working, but we still recommend to adjust the FQCNs as well.

- For community.general 2.0.0, the `postgresql` modules and plugins will be moved to the community.postgresql collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use `postgresql` content from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.postgresql.` instead of `community.general.`, for example replace `community.general.postgresql_info` in a task by `community.postgresql.postgresql_info`.

  If you use ansible-base and installed `community.general` manually and rely on the `postgresql` content, you have to make sure to install the `community.postgresql` collection as well. If you are using FQCNs, i.e. `community.general.postgresql_info` instead of `postgresql_info`, it will continue working, but we still recommend to adjust the FQCNs as well.

- The community.general collection no longer depends on the ansible.posix collection (https://github.com/ansible-collections/community.general/pull/1157).

community.network

- For community.network 2.0.0, the `routeros` modules and plugins will be moved to the community.routeros collection. A redirection will be inserted so that users using ansible-base 2.10 or newer do not have to change anything.

  If you use Ansible 2.9 and explicitly use `routeros` content from this collection, you will need to adjust your playbooks and roles to use FQCNs starting with `community.routeros.` instead of `community.network.routeros_`, for example replace `community.network.routeros_api` in a task by `community.routeros.api`.

  If you use ansible-base and installed `community.network` manually and rely on the `routeros` content, you have to make sure to install the `community.routeros` collection as well. If you are using FQCNs, i.e. `community.network.routeros_command` instead of `routeros_command`, it will continue working, but we still recommend to adjust the FQCNs as well.

- In community.network 2.0.0, the `fortimanager` httpapi plugin will be removed and replaced by a redirect to the corresponding plugin in the fortios.fortimanager collection. For Ansible 2.10 and ansible-base 2.10 users, this means that it will continue to work assuming that collection is installed. For Ansible 2.9 users, this means that they have to adjust the FQCN from `community.network.fortimanager` to `fortios.fortimanager.fortimanager` (https://github.com/ansible-collections/community.network/pull/151).

community.okd

- Add custom k8s module, integrate better Molecule tests (https://github.com/ansible-collections/community.okd/pull/7).
- Add downstream build scripts to build redhat.openshift (https://github.com/ansible-collections/community.okd/pull/20).
- Add openshift connection plugin, update inventory plugin to use it (https://github.com/ansible-collections/community.okd/pull/18).
- Add openshift_process module for template rendering and optional application of rendered resources (https://github.com/ansible-collections/community.okd/pull/44).
- Add openshift_route module for creating routes from services (https://github.com/ansible-collections/community.okd/pull/40).
- Initial content migration from community.kubernetes (https://github.com/ansible-collections/community.okd/pull/3).
- openshift_auth - new module (migrated from k8s_auth in community.kubernetes) (https://github.com/ansible-collections/community.okd/pull/33).

Removed Features

community.docker

- docker_container - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_container - the default of `networks_cli_compatible` changed to `true` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_container - the unused option `trust_image_content` has been removed (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - `state=build` has been removed. Use `present` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `container_limits`, `dockerfile`, `http_timeout`, `nocache`, `rm`, `path`, `buildargs`, `pull` have been removed. Use the corresponding suboptions of `build` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `force` option has been removed. Use the more specific `force_*` options instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `source` option is now mandatory (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the `use_tls` option has been removed. Use `tls` and `validate_certs` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image - the default of the `build.pull` option changed to `false` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_image_facts - this alias is on longer available, use `docker_image_info` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_network - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_network - the `ipam_options` option has been removed. Use `ipam_config` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_service - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm - `state=inspect` has been removed. Use `docker_swarm_info` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `constraints` option has been removed. Use `placement.constraints` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `limit_cpu` and `limit_memory` options has been removed. Use the corresponding suboptions in `limits` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `log_driver` and `log_driver_options` options has been removed. Use the corresponding suboptions in `logging` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `reserve_cpu` and `reserve_memory` options has been removed. Use the corresponding suboptions in `reservations` instead (https://github.com/ansible-collections/community.docker/pull/1).

- docker_swarm_service - the `restart_policy`, `restart_policy_attempts`, `restart_policy_delay` and `restart_policy_window` options has been removed. Use the corresponding suboptions in `restart_config` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_swarm_service - the `update_delay`, `update_parallelism`, `update_failure_action`, `update_monitor`, `update_max_failure_ratio` and `update_order` options has been removed. Use the corresponding suboptions in `update_config` instead (https://github.com/ansible-collections/community.docker/pull/1).
- docker_volume - no longer returns `ansible_facts` (https://github.com/ansible-collections/community.docker/pull/1).
- docker_volume - the `force` option has been removed. Use `recreate` instead (https://github.com/ansible-collections/community.docker/pull/1).

Deprecated Features

community.general

- django_manage - the parameter `liveserver` relates to a no longer maintained third-party module for django. It is now deprecated, and will be remove in community.general 3.0.0 (https://github.com/ansible-collections/community.general/pull/1154).
- proxmox - the default of the new `proxmox_default_behavior` option will change from `compatibility` to `no_defaults` in community.general 4.0.0. Set the option to an explicit value to avoid a deprecation warning (https://github.com/ansible-collections/community.general/pull/850).
- proxmox_kvm - the default of the new `proxmox_default_behavior` option will change from `compatibility` to `no_defaults` in community.general 4.0.0. Set the option to an explicit value to avoid a deprecation warning (https://github.com/ansible-collections/community.general/pull/850).
- syspatch - deprecate the redundant `apply` argument (https://github.com/ansible-collections/community.general/pull/360).

community.network

- Deprecate connection=local support for network platforms using persistent framework (https://github.com/ansible-collections/community.network/pull/120).

Porting Guide for v2.10.2

Breaking Changes

Ansible-base

- ansible-galaxy login command has been removed (see https://github.com/ansible/ansible/issues/71560)

Major Changes

f5networks.f5_modules

- Add phone home Teem integration into all modules, functionality can be disabled by setting up F5_TEEM environment variable or no_f5_teem provider parameter

ovirt.ovirt

- cluster_upgrade - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/94).
- disaster_recovery - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/134).
- engine_setup - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/69).
- hosted_engine_setup - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/106).
- image_template - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/95).
- infra - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/92).
- manageiq - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/97).
- repositories - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/96).
- shutdown_env - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/112).
- vm_infra - Migrate role (https://github.com/oVirt/ovirt-ansible-collection/pull/93).

Removed Features

f5networks.f5_modules

- Removed arp_state parameter from the bigip_virtual_address module

Deprecated Features

cisco.nxos

- Deprecated *nxos_interface_ospf* in favor of *nxos_ospf_interfaces* Resource Module.

Porting Guide for v2.10.1

Major Changes

community.kubernetes

- k8s - Add support for template parameter (https://github.com/ansible-collections/community.kubernetes/pull/230).
- k8s_* - Add support for vaulted kubeconfig and src (https://github.com/ansible-collections/community.kubernetes/pull/193).

Deprecated Features

cisco.nxos

- Deprecated *nxos_smu* in favor of *nxos_rpm* module.
- The *nxos_ospf_vrf* module is deprecated by *nxos_ospfv2* and *nxos_ospfv3* Resource Modules.

Porting Guide for v2.10.0

Known Issues

- Due to a limitation in pip, you cannot `pip install --upgrade` from ansible-2.9 or earlier to ansible-2.10 or higher. Instead, you must explicitly use `pip uninstall ansible` before pip installing the new version. If you attempt to upgrade Ansible with pip without first uninstalling, the installer warns you to uninstall first.
- The individual collections that make up the ansible-2.10.0 package can be viewed independently. However, they are not currently listed by ansible-galaxy. To view these collections with ansible-galaxy, explicitly specify where ansible has installed the collections – `COLLECTION_INSTALL=$(python -c 'import ansible, os.path ; print("%s/../ansible_collections" % os.path.dirname(ansible.__file__))') ansible-galaxy collection list -p "$COLLECTION_INSTALL"`.
- These fortios modules are not automatically redirected from their 2.9.x names to the new 2.10.x names within collections. You must modify your playbooks to use fully qualified collection names for them. You can use the documentation (https://docs.ansible.com/ansible/2.10/collections/fortinet/fortios/) for the `fortinet.fortios` collection to determine what the fully qualified collection names are.
  - fortios_address
  - fortios_config
  - fortios_firewall_DoS_policy
  - fortios_firewall_DoS_policy6
  - fortios_ipv4_policy
  - fortios_switch_controller_802_1X_settings
  - fortios_switch_controller_security_policy_802_1X
  - fortios_system_firmware_upgrade
  - fortios_system_nd_proxy
  - fortios_webfilter

community.grafana

- grafana_datasource doesn't set password correctly (#113)

Breaking Changes

- cisco.nxos.nxos_igmp_interface - no longer supports the deprecated `oif_prefix` and `oif_source` options. These have been superseded by `oif_ps`.
- community.grafana.grafana_dashboard - the parameter `message` is renamed to `commit_message` since `message` is used by Ansible Core engine internally.
- purestorage.flashblade.purefb_fs - no longer supports the deprecated `nfs` option. This has been superseded by `nfsv3`.

amazon.aws

- aws_s3 - can now delete versioned buckets even when they are not empty - set mode to delete to delete a versioned bucket and everything in it.

ansible.windows

- setup - Make sure `ansible_date_time.epoch` is seconds since EPOCH in UTC to mirror the POSIX facts. The `ansible_date_time.epoch_local` contains seconds since EPOCH in the local timezone for backwards compatibility
- setup - Will now add the IPv6 scope on link local addresses for `ansible_ip_addresses`
- setup - `ansible_processor` will now return the index before the other values to match the POSIX fact behaviour
- win_find - No longer filters by size on directories, this feature had a lot of bugs, slowed down the module, and not a supported scenario with the `find` module.
- **win_find - module has been refactored to better match the behaviour of the `find` module. Here is what has changed:**
  - When the directory specified by `paths` does not exist or is a file, it will no longer fail and will just warn the user
  - Junction points are no longer reported as `islnk`, use `isjunction` to properly report these files. This behaviour matches the win_stat module
  - Directories no longer return a `size`, this matches the `stat` and `find` behaviour and has been removed due to the difficulties in correctly reporting the size of a directory
- win_user - Change idempotency checks for `description` to be case sensitive
- win_user - Change idempotency checks for `fullname` to be case sensitive

cisco.meraki

- meraki_device - Changed tags from string to list
- meraki_device - Removed serial_lldp_cdp parameter
- meraki_device - Removed serial_uplink parameter
- meraki_intrusion_prevention - Rename whitedlisted_rules to allowed_rules
- meraki_mx_l3_firewall - Rule responses are now in a *rules* list
- meraki_mx_l7_firewall - Rename blacklisted_countries to blocked_countries

- meraki_mx_l7_firewall - Rename whitelisted_countries to allowed_countries
- meraki_network - Local and remote status page settings cannot be set during network creation
- meraki_network - *disableRemoteStatusPage* response is now *remote_status_page_enabled*
- meraki_network - *disable_my_meraki_com* response is now *local_status_page_enabled*
- meraki_network - *disable_my_meraki* has been deprecated
- meraki_network - *enable_my_meraki* is now called *local_status_page_enabled*
- meraki_network - *enable_remote_status_page* is now called *remote_status_page_enabled*
- meraki_network - *enabled* response for VLAN status is now *vlans_enabled*
- meraki_network - *tags* and *type* now return a list
- meraki_snmp - peer_ips is now a list
- meraki_switchport - *access_policy_number* is now an int and not a string
- meraki_switchport - *tags* is now a list and not a string
- meraki_webhook - Querying test status now uses state of query.

community.general

- The environment variable for the auth context for the oc.py connection plugin has been corrected (K8S_CONTEXT). It was using an initial lowercase k by mistake. (https://github.com/ansible-collections/community.general/pull/377).
- bigpanda - the parameter `message` was renamed to `deployment_message` since `message` is used by Ansible Core engine internally.
- cisco_spark - the module option `message` was renamed to `msg`, as `message` is used internally in Ansible Core engine (https://github.com/ansible/ansible/issues/39295)
- datadog - the parameter `message` was renamed to `notification_message` since `message` is used by Ansible Core engine internally.
- docker_container - no longer passes information on non-anonymous volumes or binds as `Volumes` to the Docker daemon. This increases compatibility with the `docker` CLI program. Note that if you specify `volumes: strict` in `comparisons`, this could cause existing containers created with docker_container from Ansible 2.9 or earlier to restart.
- docker_container - support for port ranges was adjusted to be more compatible to the `docker` command line utility: a one-port container range combined with a multiple-port host range will no longer result in only the first host port be used, but the whole range being passed to Docker so that a free port in that range will be used.
- hashi_vault lookup - now returns the latest version when using the KV v2 secrets engine. Previously, it returned all versions of the secret which required additional steps to extract and filter the desired version.
- log_plays callback - add missing information to the logs generated by the callback plugin. This changes the log message format (https://github.com/ansible-collections/community.general/pull/442).
- pkgng - passing `name: *` with `state: absent` will no longer remove every installed package from the system. It is now a noop. (https://github.com/ansible-collections/community.general/pull/569).
- pkgng - passing `name: *` with `state: latest` or `state: present` will no longer install every package from the configured package repositories. Instead, `name: *, state: latest` will upgrade all already-installed packages, and `name: *, state: present` is a noop. (https://github.com/ansible-collections/community.general/pull/569).

community.network

- routeros_facts - allow multiple addresses and neighbors per interface. This makes `ansible_net_neighbors` a list instead of a dict (https://github.com/ansible-collections/community.network/pull/6).

community.vmware

- vmware_datastore_maintenancemode - now returns `datastore_status` instead of Ansible internal key `results`.
- vmware_guest_custom_attributes - does not require VM name which was a required parameter for releases prior to Ansible 2.10.
- vmware_guest_find - the `datacenter` option has been removed.
- vmware_host_kernel_manager - now returns `host_kernel_status` instead of Ansible internal key `results`.
- vmware_host_ntp - now returns `host_ntp_status` instead of Ansible internal key `results`.
- vmware_host_service_manager - now returns `host_service_status` instead of Ansible internal key `results`.
- vmware_tag - now returns `tag_status` instead of Ansible internal key `results`.
- vmware_vmkernel - the options `ip_address` and `subnet_mask` have been removed; use the suboptions `ip_address` and `subnet_mask` of the `network` option instead.

community.windows

- win_pester - no longer runs all `*.ps1` file in the directory specified due to it executing potentially unknown scripts. It will follow the default behaviour of only running tests for files that are like `*.tests.ps1` which is built into Pester itself.

community.zabbix

- zabbix_javagateway - options `javagateway_pidfile`, `javagateway_listenip`, `javagateway_listenport` and `javagateway_startpollers` renamed to `zabbix_javagateway_xyz` (see UPGRADE.md).

netbox.netbox

- Change `ip-addresses` key in netbox inventory plugin to `ip_addresses` (https://github.com/netbox-community/ansible_modules/issues/139)
- Changed `group` to `tenant_group` in `netbox_tenant.py` (https://github.com/netbox-community/ansible_modules/issues/9)
- Changed `role` to `prefix_role` in `netbox_prefix.py` (https://github.com/netbox-community/ansible_modules/issues/9)
- Module failures when required fields aren't provided (https://github.com/netbox-community/ansible_modules/issues/24)
- Renamed `netbox_interface` to `netbox_device_interface` (https://github.com/netbox-community/ansible_modules/issues/9)

- This version has a few breaking changes due to new namespace and collection name. I felt it necessary to change the name of the lookup plugin and inventory plugin just not to have a non descriptive namespace call to use them. Below is an example: `netbox.netbox.netbox` would be used for both inventory plugin and lookup plugin, but in different contexts so no collision will arise, but confusion will. I renamed the lookup plugin to `nb_lookup` so it will be used with the FQCN `netbox.netbox.nb_lookup`. The inventory plugin will now be called within an inventory file by `netbox.netbox.nb_inventory`
- To pass in integers through Ansible Jinja filters for a key in `data` that requires querying an endpoint is now done by making it a dictionary with an `id` key. The previous behavior was to just pass in an integer and it was converted when normalizing the data, but some people may have names that are all integers and those were being converted erroneously so we made the decision to change the method to convert to an integer for the NetBox API.

```
tasks:
  - name: Create device within NetBox with only required information
    netbox_device:
      netbox_url: http://netbox-demo.org:32768
      netbox_token: 0123456789abcdef0123456789abcdef01234567
      data:
        name: Test66
        device_type:
          id: "{{ some_jinja_variable }}"
        device_role: Core Switch
        site: Test Site
        status: Staged
      state: present
```

- `pynetbox` changed to using `requests.Session()` to manage the HTTP session which broke passing in `ssl_verify` when building the NetBox API client. This PR makes `pynetbox 5.0.4+` the new required version of *pynetbox* for the Ansible modules and lookup plugin. (https://github.com/netbox-community/ansible_modules/pull/269)

theforeman.foreman

- All modules were renamed to drop the `foreman_` and `katello_` prefixes. Additionally to the prefix removal, the following modules were further ranamed:
  - katello_upload to content_upload
  - katello_sync to repository_sync
  - katello_manifest to subscription_manifest
  - foreman_search_facts to resource_info
  - foreman_ptable to partition_table
  - foreman_model to hardware_model
  - foreman_environment to puppet_environment

Major Changes

Ansible-base

- Both ansible-doc and ansible-console's help command will error for modules and plugins whose return documentation cannot be parsed as YAML. All modules and plugins passing `ansible-test sanity --test yamllint` will not be affected by this.
- Collections may declare a list of supported/tested Ansible versions for the collection. A warning is issued if a collection does not support the Ansible version that loads it (can also be configured as silent or a fatal error). Collections that do not declare supported Ansible versions do not issue a warning/error.
- Plugin routing allows collections to declare deprecation, redirection targets, and removals for all plugin types.
- Plugins that import module_utils and other ansible namespaces that have moved to collections should continue to work unmodified.
- Routing data built into Ansible 2.10 ensures that 2.9 content should work unmodified on 2.10. Formerly included modules and plugins that were moved to collections are still accessible by their original unqualified names, so long as their destination collections are installed.
- When deprecations are done in code, they to specify a `collection_name` so that deprecation warnings can mention which collection - or ansible-base - is deprecating a feature. This affects all `Display.deprecated()` or `AnsibleModule.deprecate()` or `Ansible.Basic.Deprecate()` calls, and `removed_in_version`/`removed_at_date` or `deprecated_aliases` in module argument specs.
- ansible-test now uses a different `default` test container for Ansible Collections

amazon.aws

- ec2 module_utils - The `AWSRetry` decorator no longer catches `NotFound` exceptions by default. `NotFound` exceptions need to be explicitly added using `catch_extra_error_codes`. Some AWS modules may see an increase in transient failures due to AWS''s eventual consistency model.

ansible.netcommon

- Add libssh connection plugin and refactor network_cli (https://github.com/ansible-collections/ansible.netcommon/pull/30)

ansible.posix

- Bootstrap Collection (https://github.com/ansible-collections/ansible.posix/pull/1).

cisco.meraki

- Rewrite requests method for version 1.0 API and improved readability
- meraki_mr_rf_profile - Configure wireless RF profiles.
- meraki_mr_settings - Configure network settings for wireless.
- meraki_ms_l3_interface - New module
- meraki_ms_ospf - Configure OSPF.

community.general

- docker_container - the `network_mode` option will be set by default to the name of the first network in `networks` if at least one network is given and `networks_cli_compatible` is `true` (will be default from community.general 2.0.0 on). Set to an explicit value to avoid deprecation warnings if you specify networks and set `networks_cli_compatible` to `true`. The current default (not specifying it) is equivalent to the value `default`.
- docker_container - the module has a new option, `container_default_behavior`, whose default value will change from `compatibility` to `no_defaults`. Set to an explicit value to avoid deprecation warnings.
- gitlab_user - no longer requires `name`, `email` and `password` arguments when `state=absent`.

community.grafana

- Add changelog management for ansible 2.10 (#112)
- grafana_datasource ; adding additional_json_data param

community.kubernetes

- Add changelog and fragments and document changelog process (https://github.com/ansible-collections/community.kubernetes/pull/131).
- helm - New module for managing Helm charts (https://github.com/ansible-collections/community.kubernetes/pull/61).
- helm_info - New module for retrieving Helm chart information (https://github.com/ansible-collections/community.kubernetes/pull/61).
- helm_plugin - new module to manage Helm plugins (https://github.com/ansible-collections/community.kubernetes/pull/154).
- helm_plugin_info - new modules to gather information about Helm plugins (https://github.com/ansible-collections/community.kubernetes/pull/154).
- helm_repository - New module for managing Helm repositories (https://github.com/ansible-collections/community.kubernetes/pull/61).
- k8s - Inventory source migrated from Ansible 2.9 to Kubernetes collection.
- k8s - Lookup plugin migrated from Ansible 2.9 to Kubernetes collection.
- k8s - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_auth - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_config_resource_name - Filter plugin migrated from Ansible 2.9 to Kubernetes collection.

- k8s_exec - New module for executing commands on pods through Kubernetes API (https://github.com/ansible-collections/community.kubernetes/pull/14).
- k8s_exec - Return rc for the command executed (https://github.com/ansible-collections/community.kubernetes/pull/158).
- k8s_info - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_log - New module for retrieving pod logs (https://github.com/ansible-collections/community.kubernetes/pull/16).
- k8s_scale - Module migrated from Ansible 2.9 to Kubernetes collection.
- k8s_service - Module migrated from Ansible 2.9 to Kubernetes collection.
- kubectl - Connection plugin migrated from Ansible 2.9 to Kubernetes collection.
- openshift - Inventory source migrated from Ansible 2.9 to Kubernetes collection.

community.libvirt

- added generic libvirt inventory plugin
- removed libvirt_lxc inventory script

dellemc.os10

- New role os10_aaa - Facilitates the configuration of Authentication Authorization and Accounting (AAA), TACACS and RADIUS server.
- New role os10_acl - Facilitates the configuration of Access Control lists.
- New role os10_bfd - Facilitates the configuration of BFD global attributes.
- New role os10_bgp - Facilitates the configuration of border gateway protocol (BGP) attributes.
- New role os10_copy_config - This role pushes the backup running configuration into a OS10 device.
- New role os10_dns - Facilitates the configuration of domain name service (DNS).
- New role os10_ecmp - Facilitates the configuration of equal cost multi-path (ECMP) for IPv4.
- New role os10_fabric_summary Facilitates to get show system information of all the OS10 switches in the fabric.
- New role os10_flow_monitor Facilitates the configuration of ACL flow-based monitoring attributes.
- New role os10_image_upgrade Facilitates installation of OS10 software images.
- New role os10_interface Facilitates the configuration of interface attributes.
- New role os10_lag Facilitates the configuration of link aggregation group (LAG) attributes.
- New role os10_lldp Facilitates the configuration of link layer discovery protocol (LLDP) attributes at global and interface level.
- New role os10_logging Facilitates the configuration of global logging attributes and logging servers.
- New role os10_network_validation Facilitates validation of wiring connection, BGP neighbors, MTU between neighbors and VLT pair.
- New role os10_ntp Facilitates the configuration of network time protocol (NTP) attributes.
- New role os10_prefix_list Facilitates the configuration of IP prefix-list.

- New role os10_qos Facilitates the configuration of quality of service attributes including policy-map and class-map.
- New role os10_raguard Facilitates the configuration of IPv6 RA Guard attributes.
- New role os10_route_map Facilitates the configuration of route-map attributes.
- New role os10_snmp Facilitates the configuration of global SNMP attributes.
- New role os10_system Facilitates the configuration of hostname and hashing algorithm.
- New role os10_template The role takes the raw string input from the CLI of OS10 device, and returns a structured text in the form of a Python dictionary.
- New role os10_uplink Facilitates the configuration of uplink attributes like uplink-state group.
- New role os10_users Facilitates the configuration of global system user attributes.
- New role os10_vlan Facilitates the configuration of virtual LAN (VLAN) attributes.
- New role os10_vlt Facilitates the configuration of virtual link trunking (VLT).
- New role os10_vrf Facilitates the configuration of virtual routing and forwarding (VRF).
- New role os10_vrrp Facilitates the configuration of virtual router redundancy protocol (VRRP) attributes.
- New role os10_vxlan Facilitates the configuration of virtual extensible LAN (VXLAN) attributes.
- New role os10_xstp Facilitates the configuration of xSTP attributes.

f5networks.f5_modules

- Broke apart bigip_device_auth_radius to implement radius server configuration in bigip_device_auth_server module. Refer to module documentation for usage details
- Remove redundant parameters in f5_provider to fix disparity between documentation and module parameters

gluster.gluster

- geo_rep - Added the independent module of geo rep with other gluster modules (https://github.com/gluster/gluster-ansible-collection/pull/2).

ovirt.ovirt

- ovirt_disk - Add backup (https://github.com/oVirt/ovirt-ansible-collection/pull/57).
- ovirt_disk - Support direct upload/download (https://github.com/oVirt/ovirt-ansible-collection/pull/35).
- ovirt_host - Add ssh_port (https://github.com/oVirt/ovirt-ansible-collection/pull/60).
- ovirt_vm_os_info - Creation of module (https://github.com/oVirt/ovirt-ansible-collection/pull/26).

purestorage.flasharray

- purefa_console - manage Console Lock setting for the FlashArray
- purefa_endpoint - manage VMware protocol-endpoints on the FlashArray
- purefa_eula - sign, or resign, FlashArray EULA
- purefa_inventory - get hardware inventory information from a FlashArray
- purefa_network - manage the physical and virtual network settings on the FlashArray
- purefa_pgsched - manage protection group snapshot and replication schedules on the FlashArray
- purefa_pod - manage ActiveCluster pods in FlashArrays
- purefa_pod_replica - manage ActiveDR pod replica links in FlashArrays
- purefa_proxy - manage the phonehome HTTPS proxy setting for the FlashArray
- purefa_smis - manage SMI-S settings on the FlashArray
- purefa_subnet - manage network subnets on the FlashArray
- purefa_timeout - manage the GUI idle timeout on the FlashArray
- purefa_vlan - manage VLAN interfaces on the FlashArray
- purefa_vnc - manage VNC for installed applications on the FlashArray
- purefa_volume_tags - manage volume tags on the FlashArray

purestorage.flashblade

- purefb_alert - manage alert email settings on a FlashBlade
- purefb_bladename - manage FlashBlade name
- purefb_bucket_replica - manage bucket replica links on a FlashBlade
- purefb_connect - manage connections between FlashBlades
- purefb_dns - manage DNS settings on a FlashBlade
- purefb_fs_replica - manage filesystem replica links on a FlashBlade
- purefb_inventory - get information about the hardware inventory of a FlashBlade
- purefb_ntp - manage the NTP settings for a FlashBlade
- purefb_phonehome - manage the phone home settings for a FlashBlade
- purefb_policy - manage the filesystem snapshot policies for a FlashBlade
- purefb_proxy - manage the phone home HTTP proxy settings for a FlashBlade
- purefb_remote_cred - manage the Object Store Remote Credentials on a FlashBlade
- purefb_snmp_agent - modify the FlashBlade SNMP Agent
- purefb_snmp_mgr - manage SNMP Managers on a FlashBlade
- purefb_target - manage remote S3-capable targets for a FlashBlade
- purefb_user - manage local `pureuser` account password on a FlashBlade

Removed Features

Ansible-base

- core - remove support for `check_invalid_arguments` in `AnsibleModule`, `AzureModule` and `UTMModule`.

ansible.netcommon

- module_utils.network.common.utils.ComplexDict has been removed

ansible.windows

- win_stat - removed the deprecated `get_md55` option and `md5` return value.

community.crypto

- The `letsencrypt` module has been removed. Use `acme_certificate` instead.

community.general

- conjur_variable lookup - has been moved to the `cyberark.conjur` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/570).
- core - remove support for `check_invalid_arguments` in `UTMModule`.
- digital_ocean_* - all DigitalOcean modules have been moved to the `community.digitalocean` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/622).
- infini_* - all infinidat modules have been moved to the `infinidat.infinibox` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/607).
- logicmonitor - the module has been removed in 1.0.0 since it is unmaintained and the API used by the module has been turned off in 2017 (https://github.com/ansible-collections/community.general/issues/539, https://github.com/ansible-collections/community.general/pull/541).
- logicmonitor_facts - the module has been removed in 1.0.0 since it is unmaintained and the API used by the module has been turned off in 2017 (https://github.com/ansible-collections/community.general/issues/539, https://github.com/ansible-collections/community.general/pull/541).
- mysql_* - all MySQL modules have been moved to the `community.mysql` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/633).
- pacman - Removed deprecated `recurse` option, use `extra_args=--recursive` instead
- proxysql_* - all ProxySQL modules have been moved to the `community.proxysql` collection. A redirection is active, which will be removed in version 2.0.0 (https://github.com/ansible-collections/community.general/pull/624).

community.network

- onyx - all onyx modules and plugins have been moved to the mellanox.onyx collection. Redirects have been added that will be removed in community.network 2.0.0 (https://github.com/ansible-collections/community.network/pull/83).

community.vmware

- vmware_guest_find - Removed deprecated `datacenter` option
- vmware_portgroup - removed 'inbound_policy', and 'rolling_order' deprecated options.
- vmware_vmkernel - Removed deprecated `ip_address` option; use sub-option ip_address in the network option instead
- vmware_vmkernel - Removed deprecated `subnet_mask` option; use sub-option subnet_mask in the network option instead

community.windows

- win_disk_image - removed the deprecated return value `mount_path` in favor of `mount_paths`.
- win_psexec - removed the deprecated `extra_opts` option.

f5networks.f5_modules

- Remove _bigip_iapplx_package alias
- Remove _bigip_security_address_list alias
- Remove _bigip_security_port_list alias
- Remove _bigip_traffic_group alias
- Remove bigip_appsvcs_extension module
- Remove bigip_asm_policy module

Deprecated Features

- The vyos.vyos.vyos_static_route module has been deprecated and will be removed in a later release; use vyos.vyos.vyos_static_routes instead.

Ansible-base

- Using the DefaultCallback without the correspodning doc_fragment or copying the documentation.
- hash_behaviour - Deprecate `hash_behaviour` for future removal.
- script inventory plugin - The 'cache' option is deprecated and will be removed in 2.12. Its use has been removed from the plugin since it has never had any effect.

amazon.aws

- All AWS Modules - aws_access_key, aws_secret_key and security_token will be made mutually exclusive with profile after 2022-06-01.
- cloudformation - The template_format option had no effect since Ansible 2.3 and will be removed after 2022-06-01
- cloudformation - the template_format option has been deprecated and will be removed in a later release. It has been ignored by the module since Ansible 2.3.
- data_pipeline - The version option had no effect and will be removed in after 2022-06-01
- ec2 - in a later release, the group and group_id options will become mutually exclusive. Currently group_id is ignored if you pass both.
- ec2_ami - The no_device alias NoDevice has been deprecated and will be removed after 2022-06-01
- ec2_ami - The virtual_name alias VirtualName has been deprecated and will be removed after 2022-06-01
- ec2_eip - The wait_timeout option had no effect and will be removed after 2022-06-01
- ec2_key - The wait_timeout option had no effect and will be removed after 2022-06-01
- ec2_key - The wait option had no effect and will be removed after 2022-06-01
- ec2_key - the wait_timeout option has been deprecated and will be removed in a later release. It has had no effect since Ansible 2.5.
- ec2_key - the wait option has been deprecated and will be removed in a later release. It has had no effect since Ansible 2.5.
- ec2_lc - The associate_public_ip_address option had no effect and will be removed after 2022-06-01
- ec2_tag - deprecate the list option in favor of ec2_tag_info
- ec2_tag - support for list as a state has been deprecated and will be removed in a later release. The ec2_tag_info can be used to fetch the tags on an EC2 resource.

ansible.windows

- win_domain_computer - Deprecated the undocumented log_path option. This option will be removed in a major release after 2022-07-01.
- win_domain_controller - the log_path option has been deprecated and will be removed in a later release. This was undocumented and only related to debugging information for module development.
- win_package - the ensure alias for the state option has been deprecated and will be removed in a later release. Please use state instead of ensure.
- win_package - the productid alias for the product_id option has been deprecated and will be removed in a later release. Please use product_id instead of productid.
- win_package - the username and password options has been deprecated and will be removed in a later release. The same functionality can be done by using become: yes and become_flags: logon_type=new_credentials logon_flags=netcredentials_only on the task.
- win_regedit - Deprecated using forward slashes as a path separator, use backslashes to avoid ambiguity between a forward slash in the key name or a forward slash as a path separator. This feature will be removed in a major release after 2021-07-01.

community.aws

- cloudformation - The `template_format` option had no effect since Ansible 2.3 and will be removed after 2022-06-01
- data_pipeline - The `version` option had no effect and will be removed after 2022-06-01
- data_pipeline - the `version` option has been deprecated and will be removed in a later release. It has always been ignored by the module.
- ec2_eip - The `wait_timeout` option had no effect and will be removed after 2022-06-01
- ec2_eip - the `wait_timeout` option has been deprecated and will be removed in a later release. It has had no effect since Ansible 2.3.
- ec2_key - The `wait_timeout` option had no effect and will be removed after 2022-06-01
- ec2_key - The `wait` option had no effect and will be removed after 2022-06-01
- ec2_lc - The `associate_public_ip_address` option had no effect and will be removed after 2022-06-01
- ec2_lc - the `associate_public_ip_address` option has been deprecated and will be removed after a later release. It has always been ignored by the module.
- elb_network_lb - The current default value of the `state` option has been deprecated and will change from absent to present after 2022-06-01
- elb_network_lb - in a later release, the default behaviour for the `state` option will change from `absent` to `present`. To maintain the existing behavior explicitly set state to `absent`.
- iam_managed_policy - The `fail_on_delete` option had no effect and will be removed after 2022-06-01
- iam_managed_policy - the `fail_on_delete` option has been deprecated and will be removed after a later release. It has always been ignored by the module.
- iam_policy - The `policy_document` will be removed after 2022-06-01. To maintain the existing behavior use the `policy_json` option and read the file with the `lookup` plugin.
- iam_policy - The default value of `skip_duplicates` will change after 2022-06-01 from `true` to `false`.
- iam_policy - in a later release, the default value for the `skip_duplicates` option will change from `true` to `false`. To maintain the existing behavior explicitly set it to `true`.
- iam_policy - the `policy_document` option has been deprecated and will be removed after a later release. To maintain the existing behavior use the `policy_json` option and read the file with the `lookup` plugin.
- iam_role - The default value of the purge_policies has been deprecated and will change from true to false after 2022-06-01
- iam_role - in a later release, the `purge_policies` option (also know as `purge_policy`) default value will change from `true` to `false`
- s3_lifecycle - The `requester_pays` option had no effect and will be removed after 2022-06-01
- s3_lifecycle - the `requester_pays` option has been deprecated and will be removed after a later release. It has always been ignored by the module.
- s3_sync - The `retries` option had no effect and will be removed after 2022-06-01
- s3_sync - the `retries` option has been deprecated and will be removed after 2022-06-01. It has always been ignored by the module.

community.crypto

- openssl_csr - all values for the `version` option except 1 are deprecated. The value 1 denotes the current only standardized CSR version.

community.general

- The ldap_attr module has been deprecated and will be removed in a later release; use ldap_attrs instead.
- airbrake_deployment - Add deprecation notice for `token` parameter and v2 api deploys. This feature will be removed in community.general 3.0.0.
- clc_aa_policy - The `wait` option had no effect and will be removed in community.general 3.0.0.
- clc_aa_policy - the `wait` parameter will be removed. It has always been ignored by the module.
- docker_container - the `trust_image_content` option is now deprecated and will be removed in community.general 3.0.0. It has never been used by the module.
- docker_container - the `trust_image_content` option will be removed. It has always been ignored by the module.
- docker_container - the default of `container_default_behavior` will change from `compatibility` to `no_defaults` in community.general 3.0.0. Set the option to an explicit value to avoid a deprecation warning.
- docker_container - the default value for `network_mode` will change in community.general 3.0.0, provided at least one network is specified and `networks_cli_compatible` is `true`. See porting guide, module documentation or deprecation warning for more details.
- docker_stack - Return values `out` and `err` have been deprecated and will be removed in community.general 3.0.0. Use `stdout` and `stderr` instead.
- docker_stack - the return values `err` and `out` have been deprecated. Use `stdout` and `stderr` from now on instead.
- helm - Put `helm` module to deprecated. New implementation is available in community.kubernetes collection.
- redfish_config - Deprecate `bios_attribute_name` and `bios_attribute_value` in favor of new *bios_attributes*` option.
- redfish_config - the `bios_attribute_name` and `bios_attribute_value` options will be removed. To maintain the existing behavior use the `bios_attributes` option instead.
- redfish_config and redfish_command - the behavior to select the first System, Manager, or Chassis resource to modify when multiple are present will be removed. Use the new `resource_id` option to specify target resource to modify.
- redfish_config, redfish_command - Behavior to modify the first System, Manager, or Chassis resource when multiple are present is deprecated. Use the new `resource_id` option to specify target resource to modify.
- xbps - the `force` option never had any effect. It is now deprecated, and will be removed in 3.0.0 (https://github.com/ansible-collections/community.general/pull/568).

community.vmware

- The vmware_dns_config module has been deprecated and will be removed in a later release; use vmware_host_dns instead.
- vca - vca_fw, vca_nat, vca_app are deprecated since these modules rely on deprecated part of Pyvcloud library.
- vmware_dns_config - Deprecate in favor of new module vmware_host_dns.
- vmware_guest - deprecate specifying CDROM configuration as a dict, instead use a list.
- vmware_tag_info - in a later release, the module will not return `tag_facts` since it does not return multiple tags with the same name and different category id. To maintain the existing behavior use `tag_info` which is a list of tag metadata.

community.zabbix

- zabbix_proxy (module) - deprecates `interface` sub-options `type` and `main` when proxy type is set to passive through `status=passive`. Make sure these suboptions are removed from your playbook as they were never supported by Zabbix in the first place.

f5networks.f5_modules

- Deprecated bigip_appsvcs_extension module
- Deprecated bigip_device_facts module name
- Deprecated bigiq_device_facts module name

1.3.8 Ansible 2.9 Porting Guide

This section discusses the behavioral changes between Ansible 2.8 and Ansible 2.9.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.9 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Playbook

Inventory

- `hash_behaviour` now affects inventory sources. If you have it set to `merge`, the data you get from inventory might change and you will have to update playbooks accordingly. If you're using the default setting (`overwrite`), you will see no changes. Inventory was ignoring this setting.

Loops

Ansible 2.9 handles "unsafe" data more robustly, ensuring that data marked "unsafe" is not templated. In previous versions, Ansible recursively marked all data returned by the direct use of `lookup()` as "unsafe", but only marked structured data returned by indirect lookups using `with_X` style loops as "unsafe" if the returned elements were strings. Ansible 2.9 treats these two approaches consistently.

As a result, if you use `with_dict` to return keys with templatable values, your templates may no longer work as expected in Ansible 2.9.

To allow the old behavior, switch from using `with_X` to using `loop` with a filter as described at *Migrating from with_X to loop*.

Command Line

- The location of the Galaxy token file has changed from `~/.ansible_galaxy` to `~/.ansible/galaxy_token`. You can configure both path and file name with the *GALAXY_TOKEN_PATH* config.

Deprecated

No notable changes

Collection loader changes

The way to import a PowerShell or C# module util from a collection has changed in the Ansible 2.9 release. In Ansible 2.8 a util was imported with the following syntax:

```
#AnsibleRequires -CSharpUtil AnsibleCollections.namespace_name.collection_name.util_
↪filename
#AnsibleRequires -PowerShell AnsibleCollections.namespace_name.collection_name.util_
↪filename
```

In Ansible 2.9 this was changed to:

```
#AnsibleRequires -CSharpUtil ansible_collections.namespace_name.collection_name.plugins.
↪module_utils.util_filename
#AnsibleRequires -PowerShell ansible_collections.namespace_name.collection_name.plugins.
↪module_utils.util_filename
```

The change in the collection import name also requires any C# util namespaces to be updated with the newer name format. This is more verbose but is designed to make sure we avoid plugin name conflicts across separate plugin types and to standardise how imports work in PowerShell with how Python modules work.

Modules

- The `win_get_url` and `win_uri` module now sends requests with a default `User-Agent` of `ansible-httpget`. This can be changed by using the `http_agent` key.
- The `apt` module now honors `update_cache=false` while installing its own dependency and skips the cache update. Explicitly setting `update_cache=true` or omitting the param `update_cache` will result in a cache update while installing its own dependency.
- Version 2.9.12 of Ansible changed the default mode of file-based tasks to `0o600 & ~umask` when the user did not specify a `mode` parameter on file-based tasks. This was in response to a CVE report which we have reconsidered. As a result, the mode change has been reverted in 2.9.13, and mode will now default to `0o666 & ~umask` as in previous versions of Ansible.
- If you changed any tasks to specify less restrictive permissions while using 2.9.12, those changes will be unnecessary (but will do no harm) in 2.9.13.
- To avoid the issue raised in CVE-2020-1736, specify a `mode` parameter in all file-based tasks that accept it.
- `dnf` and `yum` - As of version 2.9.13, the `dnf` module (and `yum` action when it uses `dnf`) now correctly validates GPG signatures of packages (CVE-2020-14365). If you see an error such as `Failed to validate GPG signature for [package name]`, please ensure that you have imported the correct GPG key for the DNF repository and/or package you are using. One way to do this is with the `rpm_key` module. Although we discourage it, in some cases it may be necessary to disable the GPG check. This can be done by explicitly adding `disable_gpg_check: yes` in your `dnf` or `yum` task.

Renaming from `_facts` to `_info`

Ansible 2.9 renamed a lot of modules from `<something>_facts` to `<something>_info`, because the modules do not return *Ansible facts*. Ansible facts relate to a specific host. For example, the configuration of a network interface, the operating system on a unix server, and the list of packages installed on a Windows box are all Ansible facts. The renamed modules return values that are not unique to the host. For example, account information or region data for a cloud provider. Renaming these modules should provide more clarity about the types of return values each set of modules offers.

Writing modules

- Module and module_utils files can now use relative imports to include other module_utils files. This is useful for shortening long import lines, especially in collections.

  Example of using a relative import in collections:

  ```
  # File: ansible_collections/my_namespace/my_collection/plugins/modules/my_module.py
  # Old way to use an absolute import to import module_utils from the collection:
  from ansible_collections.my_namespace.my_collection.plugins.module_utils import my_
  ↪util
  # New way using a relative import:
  from ..module_utils import my_util
  ```

  Modules and module_utils shipped with Ansible can use relative imports as well but the savings are smaller:

  ```
  # File: ansible/modules/system/ping.py
  # Old way to use an absolute import to import module_utils from core:
  from ansible.module_utils.basic import AnsibleModule
  # New way using a relative import:
  from ...module_utils.basic import AnsibleModule
  ```

  Each single dot (`.`) represents one level of the tree (equivalent to `../` in filesystem relative links).

  See also:

  The Python Relative Import Docs go into more detail of how to write relative imports.

Modules removed

The following modules no longer exist:

- Apstra's `aos_*` modules. See the new modules at https://github.com/apstra.
- ec2_ami_find use ec2_ami_facts instead.
- kubernetes use k8s instead.
- nxos_ip_interface use nxos_l3_interface instead.
- nxos_portchannel use nxos_linkagg instead.
- nxos_switchport use nxos_l2_interface instead.
- oc use k8s instead.
- panos_nat_policy use panos_nat_rule instead.
- panos_security_policy use panos_security_rule instead.

- vsphere_guest use vmware_guest instead.

Deprecation notices

The following modules will be removed in Ansible 2.13. Please update update your playbooks accordingly.

- cs_instance_facts use cs_instance_info instead.
- cs_zone_facts use cs_zone_info instead.
- digital_ocean_sshkey_facts use digital_ocean_sshkey_info instead.
- eos_interface use eos_interfaces instead.
- eos_l2_interface use eos_l2_interfaces instead.
- eos_l3_interface use eos_l3_interfaces instead.
- eos_linkagg use eos_lag_interfaces instead.
- eos_lldp_interface use eos_lldp_interfaces instead.
- eos_vlan use eos_vlans instead.
- ios_interface use ios_interfaces instead.
- ios_l2_interface use ios_l2_interfaces instead.
- ios_l3_interface use ios_l3_interfaces instead.
- ios_vlan use ios_vlans instead.
- iosxr_interface use iosxr_interfaces instead.
- junos_interface use junos_interfaces instead.
- junos_l2_interface use junos_l2_interfaces instead.
- junos_l3_interface use junos_l3_interfaces instead.
- junos_linkagg use junos_lag_interfaces instead.
- junos_lldp use junos_lldp_global instead.
- junos_lldp_interface use junos_lldp_interfaces instead.
- junos_vlan use junos_vlans instead.
- lambda_facts use lambda_info instead.
- na_ontap_gather_facts use na_ontap_info instead.
- net_banner use the platform-specific [netos]_banner modules instead.
- net_interface use the new platform-specific [netos]_interfaces modules instead.
- net_l2_interface use the new platform-specific [netos]_l2_interfaces modules instead.
- net_l3_interface use the new platform-specific [netos]_l3_interfaces modules instead.
- net_linkagg use the new platform-specific [netos]_lag modules instead.
- net_lldp use the new platform-specific [netos]_lldp_global modules instead.
- net_lldp_interface use the new platform-specific [netos]_lldp_interfaces modules instead.
- net_logging use the platform-specific [netos]_logging modules instead.
- net_static_route use the platform-specific [netos]_static_route modules instead.

- net_system use the platform-specific [netos]_system modules instead.
- net_user use the platform-specific [netos]_user modules instead.
- net_vlan use the new platform-specific [netos]_vlans modules instead.
- net_vrf use the platform-specific [netos]_vrf modules instead.
- nginx_status_facts use nginx_status_info instead.
- nxos_interface use nxos_interfaces instead.
- nxos_l2_interface use nxos_l2_interfaces instead.
- nxos_l3_interface use nxos_l3_interfaces instead.
- nxos_linkagg use nxos_lag_interfaces instead.
- nxos_vlan use nxos_vlans instead.
- online_server_facts use online_server_info instead.
- online_user_facts use online_user_info instead.
- purefa_facts use purefa_info instead.
- purefb_facts use purefb_info instead.
- scaleway_image_facts use scaleway_image_info instead.
- scaleway_ip_facts use scaleway_ip_info instead.
- scaleway_organization_facts use scaleway_organization_info instead.
- scaleway_security_group_facts use scaleway_security_group_info instead.
- scaleway_server_facts use scaleway_server_info instead.
- scaleway_snapshot_facts use scaleway_snapshot_info instead.
- scaleway_volume_facts use scaleway_volume_info instead.
- vcenter_extension_facts use vcenter_extension_info instead.
- vmware_about_facts use vmware_about_info instead.
- vmware_category_facts use vmware_category_info instead.
- vmware_drs_group_facts use vmware_drs_group_info instead.
- vmware_drs_rule_facts use vmware_drs_rule_info instead.
- vmware_dvs_portgroup_facts use vmware_dvs_portgroup_info instead.
- vmware_guest_boot_facts use vmware_guest_boot_info instead.
- vmware_guest_customization_facts use vmware_guest_customization_info instead.
- vmware_guest_disk_facts use vmware_guest_disk_info instead.
- vmware_host_capability_facts use vmware_host_capability_info instead.
- vmware_host_config_facts use vmware_host_config_info instead.
- vmware_host_dns_facts use vmware_host_dns_info instead.
- vmware_host_feature_facts use vmware_host_feature_info instead.
- vmware_host_firewall_facts use vmware_host_firewall_info instead.
- vmware_host_ntp_facts use vmware_host_ntp_info instead.

- vmware_host_package_facts use vmware_host_package_info instead.
- vmware_host_service_facts use vmware_host_service_info instead.
- vmware_host_ssl_facts use vmware_host_ssl_info instead.
- vmware_host_vmhba_facts use vmware_host_vmhba_info instead.
- vmware_host_vmnic_facts use vmware_host_vmnic_info instead.
- vmware_local_role_facts use vmware_local_role_info instead.
- vmware_local_user_facts use vmware_local_user_info instead.
- vmware_portgroup_facts use vmware_portgroup_info instead.
- vmware_resource_pool_facts use vmware_resource_pool_info instead.
- vmware_target_canonical_facts use vmware_target_canonical_info instead.
- vmware_vmkernel_facts use vmware_vmkernel_info instead.
- vmware_vswitch_facts use vmware_vswitch_info instead.
- vultr_account_facts use vultr_account_info instead.
- vultr_block_storage_facts use vultr_block_storage_info instead.
- vultr_dns_domain_facts use vultr_dns_domain_info instead.
- vultr_firewall_group_facts use vultr_firewall_group_info instead.
- vultr_network_facts use vultr_network_info instead.
- vultr_os_facts use vultr_os_info instead.
- vultr_plan_facts use vultr_plan_info instead.
- vultr_region_facts use vultr_region_info instead.
- vultr_server_facts use vultr_server_info instead.
- vultr_ssh_key_facts use vultr_ssh_key_info instead.
- vultr_startup_script_facts use vultr_startup_script_info instead.
- vultr_user_facts use vultr_user_info instead.
- vyos_interface use vyos_interfaces instead.
- vyos_l3_interface use vyos_l3_interfaces instead.
- vyos_linkagg use vyos_lag_interfaces instead.
- vyos_lldp use vyos_lldp_global instead.
- vyos_lldp_interface use vyos_lldp_interfaces instead.

The following functionality will be removed in Ansible 2.12. Please update update your playbooks accordingly.

- `vmware_cluster` DRS, HA and VSAN configuration; use vmware_cluster_drs, vmware_cluster_ha and vmware_cluster_vsan instead.

The following functionality will be removed in Ansible 2.13. Please update update your playbooks accordingly.

- `openssl_certificate` deprecates the `assertonly` provider. Please see the openssl_certificate documentation examples on how to replace the provider with the openssl_certificate_info, openssl_csr_info, openssl_privatekey_info and assert modules.

For the following modules, the PyOpenSSL-based backend `pyopenssl` has been deprecated and will be removed in Ansible 2.13:

- get_certificate
- openssl_certificate
- openssl_certificate_info
- openssl_csr
- openssl_csr_info
- openssl_privatekey
- openssl_privatekey_info
- openssl_publickey

Renamed modules

The following modules have been renamed. The old name is deprecated and will be removed in Ansible 2.13. Please update update your playbooks accordingly.

- The `ali_instance_facts` module was renamed to ali_instance_info.
- The `aws_acm_facts` module was renamed to aws_acm_info.
- The `aws_az_facts` module was renamed to aws_az_info.
- The `aws_caller_facts` module was renamed to aws_caller_info.
- The `aws_kms_facts` module was renamed to aws_kms_info.
- The `aws_region_facts` module was renamed to aws_region_info.
- The `aws_s3_bucket_facts` module was renamed to aws_s3_bucket_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `aws_sgw_facts` module was renamed to aws_sgw_info.
- The `aws_waf_facts` module was renamed to aws_waf_info.
- The `azure_rm_aks_facts` module was renamed to azure_rm_aks_info.
- The `azure_rm_aksversion_facts` module was renamed to azure_rm_aksversion_info.
- The `azure_rm_applicationsecuritygroup_facts` module was renamed to azure_rm_applicationsecuritygroup_info.
- The `azure_rm_appserviceplan_facts` module was renamed to azure_rm_appserviceplan_info.
- The `azure_rm_automationaccount_facts` module was renamed to azure_rm_automationaccount_info.
- The `azure_rm_autoscale_facts` module was renamed to azure_rm_autoscale_info.
- The `azure_rm_availabilityset_facts` module was renamed to azure_rm_availabilityset_info.
- The `azure_rm_cdnendpoint_facts` module was renamed to azure_rm_cdnendpoint_info.
- The `azure_rm_cdnprofile_facts` module was renamed to azure_rm_cdnprofile_info.
- The `azure_rm_containerinstance_facts` module was renamed to azure_rm_containerinstance_info.
- The `azure_rm_containerregistry_facts` module was renamed to azure_rm_containerregistry_info.
- The `azure_rm_cosmosdbaccount_facts` module was renamed to azure_rm_cosmosdbaccount_info.

- The `azure_rm_deployment_facts` module was renamed to azure_rm_deployment_info.
- The `azure_rm_resourcegroup_facts` module was renamed to azure_rm_resourcegroup_info.
- The `bigip_device_facts` module was renamed to bigip_device_info.
- The `bigiq_device_facts` module was renamed to bigiq_device_info.
- The `cloudformation_facts` module was renamed to cloudformation_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `cloudfront_facts` module was renamed to cloudfront_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `cloudwatchlogs_log_group_facts` module was renamed to cloudwatchlogs_log_group_info.
- The `digital_ocean_account_facts` module was renamed to digital_ocean_account_info.
- The `digital_ocean_certificate_facts` module was renamed to digital_ocean_certificate_info.
- The `digital_ocean_domain_facts` module was renamed to digital_ocean_domain_info.
- The `digital_ocean_firewall_facts` module was renamed to digital_ocean_firewall_info.
- The `digital_ocean_floating_ip_facts` module was renamed to digital_ocean_floating_ip_info.
- The `digital_ocean_image_facts` module was renamed to digital_ocean_image_info.
- The `digital_ocean_load_balancer_facts` module was renamed to digital_ocean_load_balancer_info.
- The `digital_ocean_region_facts` module was renamed to digital_ocean_region_info.
- The `digital_ocean_size_facts` module was renamed to digital_ocean_size_info.
- The `digital_ocean_snapshot_facts` module was renamed to digital_ocean_snapshot_info.
- The `digital_ocean_tag_facts` module was renamed to digital_ocean_tag_info.
- The `digital_ocean_volume_facts` module was renamed to digital_ocean_volume_info.
- The `ec2_ami_facts` module was renamed to ec2_ami_info.
- The `ec2_asg_facts` module was renamed to ec2_asg_info.
- The `ec2_customer_gateway_facts` module was renamed to ec2_customer_gateway_info.
- The `ec2_eip_facts` module was renamed to ec2_eip_info.
- The `ec2_elb_facts` module was renamed to ec2_elb_info.
- The `ec2_eni_facts` module was renamed to ec2_eni_info.
- The `ec2_group_facts` module was renamed to ec2_group_info.
- The `ec2_instance_facts` module was renamed to ec2_instance_info.
- The `ec2_lc_facts` module was renamed to ec2_lc_info.
- The `ec2_placement_group_facts` module was renamed to ec2_placement_group_info.
- The `ec2_snapshot_facts` module was renamed to ec2_snapshot_info.
- The `ec2_vol_facts` module was renamed to ec2_vol_info.
- The `ec2_vpc_dhcp_option_facts` module was renamed to ec2_vpc_dhcp_option_info.
- The `ec2_vpc_endpoint_facts` module was renamed to ec2_vpc_endpoint_info.
- The `ec2_vpc_igw_facts` module was renamed to ec2_vpc_igw_info.
- The `ec2_vpc_nacl_facts` module was renamed to ec2_vpc_nacl_info.

- The `ec2_vpc_nat_gateway_facts` module was renamed to ec2_vpc_nat_gateway_info.
- The `ec2_vpc_net_facts` module was renamed to ec2_vpc_net_info.
- The `ec2_vpc_peering_facts` module was renamed to ec2_vpc_peering_info.
- The `ec2_vpc_route_table_facts` module was renamed to ec2_vpc_route_table_info.
- The `ec2_vpc_subnet_facts` module was renamed to ec2_vpc_subnet_info.
- The `ec2_vpc_vgw_facts` module was renamed to ec2_vpc_vgw_info.
- The `ec2_vpc_vpn_facts` module was renamed to ec2_vpc_vpn_info.
- The `ecs_service_facts` module was renamed to ecs_service_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ecs_taskdefinition_facts` module was renamed to ecs_taskdefinition_info.
- The `efs_facts` module was renamed to efs_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `elasticache_facts` module was renamed to elasticache_info.
- The `elb_application_lb_facts` module was renamed to elb_application_lb_info.
- The `elb_classic_lb_facts` module was renamed to elb_classic_lb_info.
- The `elb_target_facts` module was renamed to elb_target_info.
- The `elb_target_group_facts` module was renamed to elb_target_group_info.
- The `gcp_bigquery_dataset_facts` module was renamed to gcp_bigquery_dataset_info.
- The `gcp_bigquery_table_facts` module was renamed to gcp_bigquery_table_info.
- The `gcp_cloudbuild_trigger_facts` module was renamed to gcp_cloudbuild_trigger_info.
- The `gcp_compute_address_facts` module was renamed to gcp_compute_address_info.
- The `gcp_compute_backend_bucket_facts` module was renamed to gcp_compute_backend_bucket_info.
- The `gcp_compute_backend_service_facts` module was renamed to gcp_compute_backend_service_info.
- The `gcp_compute_disk_facts` module was renamed to gcp_compute_disk_info.
- The `gcp_compute_firewall_facts` module was renamed to gcp_compute_firewall_info.
- The `gcp_compute_forwarding_rule_facts` module was renamed to gcp_compute_forwarding_rule_info.
- The `gcp_compute_global_address_facts` module was renamed to gcp_compute_global_address_info.
- The `gcp_compute_global_forwarding_rule_facts` module was renamed to gcp_compute_global_forwarding_rule_info.
- The `gcp_compute_health_check_facts` module was renamed to gcp_compute_health_check_info.
- The `gcp_compute_http_health_check_facts` module was renamed to gcp_compute_http_health_check_info.
- The `gcp_compute_https_health_check_facts` module was renamed to gcp_compute_https_health_check_info.
- The `gcp_compute_image_facts` module was renamed to gcp_compute_image_info.
- The `gcp_compute_instance_facts` module was renamed to gcp_compute_instance_info.
- The `gcp_compute_instance_group_facts` module was renamed to gcp_compute_instance_group_info.

- The `gcp_compute_instance_group_manager_facts` module was renamed to gcp_compute_instance_group_manager_info.
- The `gcp_compute_instance_template_facts` module was renamed to gcp_compute_instance_template_info.
- The `gcp_compute_interconnect_attachment_facts` module was renamed to gcp_compute_interconnect_attachment_info.
- The `gcp_compute_network_facts` module was renamed to gcp_compute_network_info.
- The `gcp_compute_region_disk_facts` module was renamed to gcp_compute_region_disk_info.
- The `gcp_compute_route_facts` module was renamed to gcp_compute_route_info.
- The `gcp_compute_router_facts` module was renamed to gcp_compute_router_info.
- The `gcp_compute_ssl_certificate_facts` module was renamed to gcp_compute_ssl_certificate_info.
- The `gcp_compute_ssl_policy_facts` module was renamed to gcp_compute_ssl_policy_info.
- The `gcp_compute_subnetwork_facts` module was renamed to gcp_compute_subnetwork_info.
- The `gcp_compute_target_http_proxy_facts` module was renamed to gcp_compute_target_http_proxy_info.
- The `gcp_compute_target_https_proxy_facts` module was renamed to gcp_compute_target_https_proxy_info.
- The `gcp_compute_target_pool_facts` module was renamed to gcp_compute_target_pool_info.
- The `gcp_compute_target_ssl_proxy_facts` module was renamed to gcp_compute_target_ssl_proxy_info.
- The `gcp_compute_target_tcp_proxy_facts` module was renamed to gcp_compute_target_tcp_proxy_info.
- The `gcp_compute_target_vpn_gateway_facts` module was renamed to gcp_compute_target_vpn_gateway_info.
- The `gcp_compute_url_map_facts` module was renamed to gcp_compute_url_map_info.
- The `gcp_compute_vpn_tunnel_facts` module was renamed to gcp_compute_vpn_tunnel_info.
- The `gcp_container_cluster_facts` module was renamed to gcp_container_cluster_info.
- The `gcp_container_node_pool_facts` module was renamed to gcp_container_node_pool_info.
- The `gcp_dns_managed_zone_facts` module was renamed to gcp_dns_managed_zone_info.
- The `gcp_dns_resource_record_set_facts` module was renamed to gcp_dns_resource_record_set_info.
- The `gcp_iam_role_facts` module was renamed to gcp_iam_role_info.
- The `gcp_iam_service_account_facts` module was renamed to gcp_iam_service_account_info.
- The `gcp_pubsub_subscription_facts` module was renamed to gcp_pubsub_subscription_info.
- The `gcp_pubsub_topic_facts` module was renamed to gcp_pubsub_topic_info.
- The `gcp_redis_instance_facts` module was renamed to gcp_redis_instance_info.
- The `gcp_resourcemanager_project_facts` module was renamed to gcp_resourcemanager_project_info.
- The `gcp_sourcerepo_repository_facts` module was renamed to gcp_sourcerepo_repository_info.
- The `gcp_spanner_database_facts` module was renamed to gcp_spanner_database_info.
- The `gcp_spanner_instance_facts` module was renamed to gcp_spanner_instance_info.
- The `gcp_sql_database_facts` module was renamed to gcp_sql_database_info.

- The gcp_sql_instance_facts module was renamed to gcp_sql_instance_info.
- The gcp_sql_user_facts module was renamed to gcp_sql_user_info.
- The gcp_tpu_node_facts module was renamed to gcp_tpu_node_info.
- The gcpubsub_facts module was renamed to gcpubsub_info.
- The github_webhook_facts module was renamed to github_webhook_info.
- The gluster_heal_facts module was renamed to gluster_heal_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_datacenter_facts module was renamed to hcloud_datacenter_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_floating_ip_facts module was renamed to hcloud_floating_ip_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_image_facts module was renamed to hcloud_image_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_location_facts module was renamed to hcloud_location_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_server_facts module was renamed to hcloud_server_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_server_type_facts module was renamed to hcloud_server_type_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_ssh_key_facts module was renamed to hcloud_ssh_key_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hcloud_volume_facts module was renamed to hcloud_volume_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The hpilo_facts module was renamed to hpilo_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The iam_mfa_device_facts module was renamed to iam_mfa_device_info.
- The iam_role_facts module was renamed to iam_role_info.
- The iam_server_certificate_facts module was renamed to iam_server_certificate_info.
- The idrac_redfish_facts module was renamed to idrac_redfish_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The intersight_facts module was renamed to intersight_info.
- The jenkins_job_facts module was renamed to jenkins_job_info.
- The k8s_facts module was renamed to k8s_info.
- The memset_memstore_facts module was renamed to memset_memstore_info.
- The memset_server_facts module was renamed to memset_server_info.
- The one_image_facts module was renamed to one_image_info.
- The onepassword_facts module was renamed to onepassword_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.
- The oneview_datacenter_facts module was renamed to oneview_datacenter_info. When called with the new name, the module no longer returns ansible_facts. To access return values, *register a variable*.

- The `oneview_enclosure_facts` module was renamed to oneview_enclosure_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_ethernet_network_facts` module was renamed to oneview_ethernet_network_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_fc_network_facts` module was renamed to oneview_fc_network_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_fcoe_network_facts` module was renamed to oneview_fcoe_network_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_logical_interconnect_group_facts` module was renamed to oneview_logical_interconnect_group_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_network_set_facts` module was renamed to oneview_network_set_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `oneview_san_manager_facts` module was renamed to oneview_san_manager_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_flavor_facts` module was renamed to os_flavor_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_image_facts` module was renamed to os_image_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_keystone_domain_facts` module was renamed to os_keystone_domain_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_networks_facts` module was renamed to os_networks_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_port_facts` module was renamed to os_port_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_project_facts` module was renamed to os_project_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_server_facts` module was renamed to os_server_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_subnets_facts` module was renamed to os_subnets_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `os_user_facts` module was renamed to os_user_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_affinity_label_facts` module was renamed to ovirt_affinity_label_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_api_facts` module was renamed to ovirt_api_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_cluster_facts` module was renamed to ovirt_cluster_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_datacenter_facts` module was renamed to ovirt_datacenter_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_disk_facts` module was renamed to ovirt_disk_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.

- The `ovirt_event_facts` module was renamed to ovirt_event_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_external_provider_facts` module was renamed to ovirt_external_provider_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_group_facts` module was renamed to ovirt_group_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_host_facts` module was renamed to ovirt_host_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_host_storage_facts` module was renamed to ovirt_host_storage_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_network_facts` module was renamed to ovirt_network_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_nic_facts` module was renamed to ovirt_nic_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_permission_facts` module was renamed to ovirt_permission_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_quota_facts` module was renamed to ovirt_quota_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_scheduling_policy_facts` module was renamed to ovirt_scheduling_policy_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_snapshot_facts` module was renamed to ovirt_snapshot_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_storage_domain_facts` module was renamed to ovirt_storage_domain_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_storage_template_facts` module was renamed to ovirt_storage_template_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_storage_vm_facts` module was renamed to ovirt_storage_vm_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_tag_facts` module was renamed to ovirt_tag_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_template_facts` module was renamed to ovirt_template_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_user_facts` module was renamed to ovirt_user_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_vm_facts` module was renamed to ovirt_vm_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `ovirt_vmpool_facts` module was renamed to ovirt_vmpool_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `python_requirements_facts` module was renamed to python_requirements_info.
- The `rds_instance_facts` module was renamed to rds_instance_info.
- The `rds_snapshot_facts` module was renamed to rds_snapshot_info.
- The `redfish_facts` module was renamed to redfish_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.

- The `redshift_facts` module was renamed to redshift_info.
- The `route53_facts` module was renamed to route53_info.
- The `smartos_image_facts` module was renamed to smartos_image_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `vertica_facts` module was renamed to vertica_info. When called with the new name, the module no longer returns `ansible_facts`. To access return values, *register a variable*.
- The `vmware_cluster_facts` module was renamed to vmware_cluster_info.
- The `vmware_datastore_facts` module was renamed to vmware_datastore_info.
- The `vmware_guest_facts` module was renamed to vmware_guest_info.
- The `vmware_guest_snapshot_facts` module was renamed to vmware_guest_snapshot_info.
- The `vmware_tag_facts` module was renamed to vmware_tag_info.
- The `vmware_vm_facts` module was renamed to vmware_vm_info.
- The `xenserver_guest_facts` module was renamed to xenserver_guest_info.
- The `zabbix_group_facts` module was renamed to zabbix_group_info.
- The `zabbix_host_facts` module was renamed to zabbix_host_info.

Noteworthy module changes

- vmware_cluster was refactored for easier maintenance/bugfixes. Use the three new, specialized modules to configure clusters. Configure DRS with vmware_cluster_drs, HA with vmware_cluster_ha and vSAN with vmware_cluster_vsan.
- vmware_dvswitch accepts `folder` parameter to place dvswitch in user defined folder. This option makes `datacenter` as an optional parameter.
- vmware_datastore_cluster accepts `folder` parameter to place datastore cluster in user defined folder. This option makes `datacenter` as an optional parameter.
- mysql_db returns new `db_list` parameter in addition to `db` parameter. This `db_list` parameter refers to list of database names. `db` parameter will be deprecated in version 2.13.
- snow_record and snow_record_find now takes environment variables for `instance`, `username` and `password` parameters. This change marks these parameters as optional.
- The deprecated `force` option in `win_firewall_rule` has been removed.
- openssl_certificate's `ownca` provider creates authority key identifiers if not explicitly disabled with `ownca_create_authority_key_identifier: no`. This is only the case for the `cryptography` backend, which is selected by default if the `cryptography` library is available.
- openssl_certificate's `ownca` and `selfsigned` providers create subject key identifiers if not explicitly disabled with `ownca_create_subject_key_identifier: never_create` resp. `selfsigned_create_subject_key_identifier: never_create`. If a subject key identifier is provided by the CSR, it is taken; if not, it is created from the public key. This is only the case for the `cryptography` backend, which is selected by default if the `cryptography` library is available.
- openssh_keypair now applies the same file permissions and ownership to both public and private keys (both get the same `mode`, `owner`, `group`, and so on). If you need to change permissions / ownership on one key, use the file to modify it after it is created.

Plugins

Removed Lookup Plugins

- redis_kv use redis instead.

Porting custom scripts

No notable changes

Networking

Network resource modules

Ansible 2.9 introduced the first batch of network resource modules. Sections of a network device's configuration can be thought of as a resource provided by that device. Network resource modules are intentionally scoped to configure a single resource and you can combine them as building blocks to configure complex network services. The older modules are deprecated in Ansible 2.9 and will be removed in Ansible 2.13. You should scan the list of deprecated modules above and replace them with the new network resource modules in your playbooks. See Ansible Network Features in 2.9 for details.

Improved gather_facts support for network devices

In Ansible 2.9, the gather_facts keyword now supports gathering network device facts in standardized key/value pairs. You can feed these network facts into further tasks to manage the network device. You can also use the new gather_network_resources parameter with the network *_facts modules (such as eos_facts) to return just a subset of the device configuration. See *Gathering facts from network devices* for an example.

Top-level connection arguments removed in 2.9

Top-level connection arguments like username, host, and password are removed in version 2.9.

OLD In Ansible < 2.4

```
- name: example of using top-level options for connection properties
  ios_command:
    commands: show version
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    authorize: yes
    auth_pass: cisco
```

Change your playbooks to the connection types network_cli and netconf using standard Ansible connection properties, and setting those properties in inventory by group. As you update your playbooks and inventory files, you can easily make the change to become for privilege escalation (on platforms that support it). For more information, see the *using become with network modules* guide and the *platform documentation*.

1.3.9 Ansible 2.8 Porting Guide

This section discusses the behavioral changes between Ansible 2.7 and Ansible 2.8.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.8 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Playbook

Distribution Facts

The information returned for the `ansible_distribution_*` group of facts may have changed slightly. Ansible 2.8 uses a new backend library for information about distributions: nir0s/distro. This library runs on Python-3.8 and fixes many bugs, including correcting release and version names.

The two facts used in playbooks most often, `ansible_distribution` and `ansible_distribution_major_version`, should not change. If you discover a change in these facts, please file a bug so we can address the difference. However, other facts like `ansible_distribution_release` and `ansible_distribution_version` may change as erroneous information gets corrected.

Imports as handlers

Beginning in version 2.8, a task cannot notify `import_tasks` or a static `include` that is specified in `handlers`.

The goal of a static import is to act as a pre-processor, where the import is replaced by the tasks defined within the imported file. When using an import, a task can notify any of the named tasks within the imported file, but not the name of the import itself.

To achieve the results of notifying a single name but running multiple handlers, utilize `include_tasks`, or `listen` *Handlers: running operations on change*.

Jinja Undefined values

Beginning in version 2.8, attempting to access an attribute of an Undefined value in Jinja will return another Undefined value, rather than throwing an error immediately. This means that you can now simply use a default with a value in a nested data structure when you don't know if the intermediate values are defined.

In Ansible 2.8:

```
{{ foo.bar.baz | default('DEFAULT') }}
```

In Ansible 2.7 and older:

```
{{ ((foo | default({})).bar | default({})).baz | default('DEFAULT') }}
```

or:

```
{{ foo.bar.baz if (foo is defined and foo.bar is defined and foo.bar.baz is defined)␣
↪else 'DEFAULT' }}
```

Module option conversion to string

Beginning in version 2.8, Ansible will warn if a module expects a string, but a non-string value is passed and automatically converted to a string. This highlights potential problems where, for example, a `yes` or `true` (parsed as truish boolean value) would be converted to the string `'True'`, or where a version number `1.10` (parsed as float value) would be converted to `'1.1'`. Such conversions can result in unexpected behavior depending on context.

This behavior can be changed to be an error or to be ignored by setting the `ANSIBLE_STRING_CONVERSION_ACTION` environment variable, or by setting the `string_conversion_action` configuration in the `defaults` section of `ansible.cfg`.

Command line facts

`cmdline` facts returned in system will be deprecated in favor of `proc_cmdline`. This change handles special case where Kernel command line parameter contains multiple values with the same key.

Bare variables in conditionals

In Ansible 2.7 and earlier, top-level variables sometimes treated boolean strings as if they were boolean values. This led to inconsistent behavior in conditional tests built on top-level variables defined as strings. Ansible 2.8 began changing this behavior. For example, if you set two conditions like this:

```
tasks:
  - include_tasks: teardown.yml
    when: teardown

  - include_tasks: provision.yml
    when: not teardown
```

based on a variable you define **as a string** (with quotation marks around it):

- In Ansible 2.7 and earlier, the two conditions above evaluated as `True` and `False` respectively if `teardown: 'true'`
- In Ansible 2.7 and earlier, both conditions evaluated as `False` if `teardown:  'false'`
- In Ansible 2.8 and later, you have the option of disabling conditional bare variables, so `when:  teardown` always evaluates as `True` and `when:  not teardown` always evaluates as `False` when `teardown` is a non-empty string (including `'true'` or `'false'`)

Ultimately, `when:  'string'` will always evaluate as `True` and `when:  not 'string'` will always evaluate as `False`, as long as `'string'` is not empty, even if the value of `'string'` itself looks like a boolean. For users with playbooks that depend on the old behavior, we added a config setting that preserves it. You can use the `ANSIBLE_CONDITIONAL_BARE_VARS` environment variable or `conditional_bare_variables` in the `defaults` section of `ansible.cfg` to select the behavior you want on your control node. The default setting is `true`, which preserves the old behavior. Set the config value or environment variable to `false` to start using the new option.

Note: In 2.10 the default setting for `conditional_bare_variables` will change to `false`. In 2.12 the old behavior will be deprecated.

Updating your playbooks

To prepare your playbooks for the new behavior, you must update your conditional statements so they accept only boolean values. For variables, you can use the `bool` filter to evaluate the string `'false'` as `False`:

```
vars:
  teardown: 'false'

tasks:
  - include_tasks: teardown.yml
    when: teardown | bool
```

(continues on next page)

(continued from previous page)

```
  - include_tasks: provision.yml
    when: not teardown | bool
```

Alternatively, you can re-define your variables as boolean values (without quotation marks) instead of strings:

```
vars:
  teardown: false

tasks:
  - include_tasks: teardown.yml
    when: teardown

  - include_tasks: provision.yml
    when: not teardown
```

For dictionaries and lists, use the `length` filter to evaluate the presence of a dictionary or list as `True`:

```
- debug:
  when: my_list | length > 0

- debug:
  when: my_dictionary | length > 0
```

Do not use the `bool` filter with lists or dictionaries. If you use `bool` with a list or dict, Ansible will always evaluate it as `False`.

Double-interpolation

The `conditional_bare_variables` setting also affects variables set based on other variables. The old behavior unexpectedly double-interpolated those variables. For example:

```
vars:
  double_interpolated: 'bare_variable'
  bare_variable: false

tasks:
  - debug:
    when: double_interpolated
```

- In Ansible 2.7 and earlier, `when:  double_interpolated` evaluated to the value of `bare_variable`, in this case, `False`. If the variable `bare_variable` is undefined, the conditional fails.
- In Ansible 2.8 and later, with bare variables disabled, Ansible evaluates `double_interpolated` as the string `'bare_variable'`, which is `True`.

To double-interpolate variable values, use curly braces:

```
vars:
  double_interpolated: "{{ other_variable }}"
  other_variable: false
```

Nested variables

The `conditional_bare_variables` setting does not affect nested variables. Any string value assigned to a subkey is already respected and not treated as a boolean. If `complex_variable['subkey']` is a non-empty string, then `when: complex_variable['subkey']` is always `True` and `when: not complex_variable['subkey']` is always `False`. If you want a string subkey like `complex_variable['subkey']` to be evaluated as a boolean, you must use the `bool` filter.

Gathering Facts

In Ansible 2.8 the implicit "Gathering Facts" task in a play was changed to obey play tags. Previous to 2.8, the "Gathering Facts" task would ignore play tags and tags supplied from the command line and always run in a task.

The behavior change affects the following example play.

```
- name: Configure Webservers
  hosts: webserver
  tags:
    - webserver
  tasks:
    - name: Install nginx
      package:
        name: nginx
      tags:
        - nginx
```

In Ansible 2.8, if you supply `--tags nginx`, the implicit "Gathering Facts" task will be skipped, as the task now inherits the tag of `webserver` instead of `always`.

If no play level tags are set, the "Gathering Facts" task will be given a tag of `always` and will effectively match prior behavior.

You can achieve similar results to the pre-2.8 behavior, by using an explicit `gather_facts` task in your `tasks` list.

```
- name: Configure Webservers
  hosts: webserver
  gather_facts: false
  tags:
    - webserver
  tasks:
    - name: Gathering Facts
      gather_facts:
      tags:
        - always

    - name: Install nginx
      package:
        name: nginx
      tags:
        - nginx
```

Python Interpreter Discovery

In Ansible 2.7 and earlier, Ansible defaulted to **/usr/bin/python** as the setting for `ansible_python_interpreter`. If you ran Ansible against a system that installed Python with a different name or a different path, your playbooks would fail with `/usr/bin/python: bad interpreter: No such file or directory` unless you either set `ansible_python_interpreter` to the correct value for that system or added a Python interpreter and any necessary dependencies at **usr/bin/python**.

Starting in Ansible 2.8, Ansible searches for the correct path and executable name for Python on each target system, first in a lookup table of default Python interpreters for common distros, then in an ordered fallback list of possible Python interpreter names/paths.

It's risky to rely on a Python interpreter set from the fallback list, because the interpreter may change on future runs. If an interpreter from higher in the fallback list gets installed (for example, as a side-effect of installing other packages), your original interpreter and its dependencies will no longer be used. For this reason, Ansible warns you when it uses a Python interpreter discovered from the fallback list. If you see this warning, the best solution is to explicitly set `ansible_python_interpreter` to the path of the correct interpreter for those target systems.

You can still set `ansible_python_interpreter` to a specific path at any variable level (as a host variable, in vars files, in playbooks, and so on). If you prefer to use the Python interpreter discovery behavior, use one of the four new values for `ansible_python_interpreter` introduced in Ansible 2.8:

New value	Behavior
auto (future default)	If a Python interpreter is discovered, Ansible uses the discovered Python, even if **/usr/bin/python** is also present. Warns when using the fallback list.
auto_legac (Ansible 2.8 default)	If a Python interpreter is discovered, and **/usr/bin/python** is absent, Ansible uses the discovered Python. Warns when using the fallback list. If a Python interpreter is discovered, and **/usr/bin/python** is present, Ansible uses **/usr/bin/python** and prints a deprecation warning about future default behavior. Warns when using the fallback list.
auto_legacy	Behaves like `auto_legacy` but suppresses the deprecation and fallback-list warnings.
auto_silent	Behaves like `auto` but suppresses the fallback-list warning.

In Ansible 2.12, Ansible will switch the default from `auto_legacy` to `auto`. The difference in behaviour is that `auto_legacy` uses **/usr/bin/python** if present and falls back to the discovered Python when it is not present. `auto` will always use the discovered Python, regardless of whether **/usr/bin/python** exists. The `auto_legacy` setting provides compatibility with previous versions of Ansible that always defaulted to **/usr/bin/python**.

If you installed Python and dependencies (`boto`, and so on) to **/usr/bin/python** as a workaround on distros with a different default Python interpreter (for example, Ubuntu 16.04+, RHEL8, Fedora 23+), you have two options:

1. Move existing dependencies over to the default Python for each platform/distribution/version.
2. Use `auto_legacy`. This setting lets Ansible find and use the workaround Python on hosts that have it, while also finding the correct default Python on newer hosts. But remember, the default will change in 4 releases.

Retry File Creation default

In Ansible 2.8, `retry_files_enabled` now defaults to `False` instead of `True`. The behavior can be modified to previous version by editing the default `ansible.cfg` file and setting the value to `True`.

Command Line

Become Prompting

Beginning in version 2.8, by default Ansible will use the word `BECOME` to prompt you for a password for elevated privileges (`sudo` privileges on Unix systems or `enable` mode on network devices):

By default in Ansible 2.8:

```
ansible-playbook --become --ask-become-pass site.yml
BECOME password:
```

If you want the prompt to display the specific `become_method` you're using, instead of the general value `BECOME`, set *AGNOSTIC_BECOME_PROMPT* to `False` in your Ansible configuration.

By default in Ansible 2.7, or with `AGNOSTIC_BECOME_PROMPT=False` in Ansible 2.8:

```
ansible-playbook --become --ask-become-pass site.yml
SUDO password:
```

Deprecated

- Setting the async directory using `ANSIBLE_ASYNC_DIR` as an task/play environment key is deprecated and will be removed in Ansible 2.12. You can achieve the same result by setting `ansible_async_dir` as a variable like:

  ```
  - name: run task with custom async directory
    command: sleep 5
    async: 10
    vars:
      ansible_async_dir: /tmp/.ansible_async
  ```

- Plugin writers who need a `FactCache` object should be aware of two deprecations:

  1. The `FactCache` class has moved from `ansible.plugins.cache.FactCache` to `ansible.vars.fact_cache.FactCache`. This is because the `FactCache` is not part of the cache plugin API and cache plugin authors should not be subclassing it. `FactCache` is still available from its old location but will issue a deprecation warning when used from there. The old location will be removed in Ansible 2.12.

  2. The `FactCache.update()` method has been converted to follow the dict API. It now takes a dictionary as its sole argument and updates itself with the dictionary's items. The previous API where `update()` took a key and a value will now issue a deprecation warning and will be removed in 2.12. If you need the old behavior switch to `FactCache.first_order_merge()` instead.

- Supporting file-backed caching through self.cache is deprecated and will be removed in Ansible 2.12. If you maintain an inventory plugin, update it to use `self._cache` as a dictionary. For implementation details, see the *developer guide on inventory plugins*.

- Importing cache plugins directly is deprecated and will be removed in Ansible 2.12. Use the plugin_loader so direct options, environment variables, and other means of configuration can be reconciled using the config system rather than constants.

```
from ansible.plugins.loader import cache_loader
cache = cache_loader.get('redis', **kwargs)
```

Modules

Major changes in popular modules are detailed here

The exec wrapper that runs PowerShell modules has been changed to set `$ErrorActionPreference = "Stop"` globally. This may mean that custom modules can fail if they implicitly relied on this behavior. To get the old behavior back, add `$ErrorActionPreference = "Continue"` to the top of the module. This change was made to restore the old behavior of the EAP that was accidentally removed in a previous release and ensure that modules are more resilient to errors that may occur in execution.

- Version 2.8.14 of Ansible changed the default mode of file-based tasks to `0o600 & ~umask` when the user did not specify a `mode` parameter on file-based tasks. This was in response to a CVE report which we have reconsidered. As a result, the `mode` change has been reverted in 2.8.15, and `mode` will now default to `0o666 & ~umask` as in previous versions of Ansible.
- If you changed any tasks to specify less restrictive permissions while using 2.8.14, those changes will be unnecessary (but will do no harm) in 2.8.15.
- To avoid the issue raised in CVE-2020-1736, specify a `mode` parameter in all file-based tasks that accept it.
- `dnf` and `yum` - As of version 2.8.15, the `dnf` module (and `yum` action when it uses `dnf`) now correctly validates GPG signatures of packages (CVE-2020-14365). If you see an error such as `Failed to validate GPG signature for [package name]`, please ensure that you have imported the correct GPG key for the DNF repository and/or package you are using. One way to do this is with the `rpm_key` module. Although we discourage it, in some cases it may be necessary to disable the GPG check. This can be done by explicitly adding `disable_gpg_check: yes` in your `dnf` or `yum` task.

Modules removed

The following modules no longer exist:

- ec2_remote_facts
- azure
- cs_nic
- netscaler
- win_msi

Deprecation notices

The following modules will be removed in Ansible 2.12. Please update your playbooks accordingly.

- `foreman` use foreman-ansible-modules instead.
- `katello` use foreman-ansible-modules instead.
- `github_hooks` use github_webhook and github_webhook_facts instead.
- `digital_ocean` use digital_ocean_droplet instead.
- `gce` use gcp_compute_instance instead.

- `gcspanner` use gcp_spanner_instance and gcp_spanner_database instead.
- `gcdns_record` use gcp_dns_resource_record_set instead.
- `gcdns_zone` use gcp_dns_managed_zone instead.
- `gcp_forwarding_rule` use gcp_compute_global_forwarding_rule or gcp_compute_forwarding_rule instead.
- `gcp_healthcheck` use gcp_compute_health_check, gcp_compute_http_health_check, or gcp_compute_https_health_check instead.
- `gcp_backend_service` use gcp_compute_backend_service instead.
- `gcp_target_proxy` use gcp_compute_target_http_proxy instead.
- `gcp_url_map` use gcp_compute_url_map instead.
- `panos` use the Palo Alto Networks Ansible Galaxy role instead.

Noteworthy module changes

- The `foreman` and `katello` modules have been deprecated in favor of a set of modules that are broken out per entity with better idempotency in mind.
- The `foreman` and `katello` modules replacement is officially part of the Foreman Community and supported there.
- The `tower_credential` module originally required the `ssh_key_data` to be the path to a ssh_key_file. In order to work like AWX/Tower/RHAAP, `ssh_key_data` now contains the content of the file. The previous behavior can be achieved with `lookup('file', '/path/to/file')`.
- The `win_scheduled_task` module deprecated support for specifying a trigger repetition as a list and this format will be removed in Ansible 2.12. Instead specify the repetition as a dictionary value.
- The `win_feature` module has removed the deprecated `restart_needed` return value, use the standardized `reboot_required` value instead.
- The `win_package` module has removed the deprecated `restart_required` and `exit_code` return value, use the standardized `reboot_required` and `rc` value instead.
- The `win_get_url` module has removed the deprecated `win_get_url` return dictionary, contained values are returned directly.
- The `win_get_url` module has removed the deprecated `skip_certificate_validation` option, use the standardized `validate_certs` option instead.
- The `vmware_local_role_facts` module now returns a list of dicts instead of a dict of dicts for role information.
- If `docker_network` or `docker_volume` were called with `diff: yes`, `check_mode: yes` or `debug: yes`, a return value called `diff` was returned of type `list`. To enable proper diff output, this was changed to type `dict`; the original `list` is returned as `diff.differences`.
- The `na_ontap_cluster_peer` module has replaced `source_intercluster_lif` and `dest_intercluster_lif` string options with `source_intercluster_lifs` and `dest_intercluster_lifs` list options
- The `modprobe` module now detects kernel builtins. Previously, attempting to remove (with `state: absent`) a builtin kernel module succeeded without any error message because `modprobe` did not detect the module as `present`. Now, `modprobe` will fail if a kernel module is builtin and `state: absent` (with an error message from the modprobe binary like `modprobe: ERROR: Module nfs is builtin.`), and it will succeed without reporting changed if `state: present`. Any playbooks that are using `changed_when: no` to mask this quirk

can safely remove that workaround. To get the previous behavior when applying `state: absent` to a builtin kernel module, use `failed_when: false` or `ignore_errors: true` in your playbook.

- The `digital_ocean` module has been deprecated in favor of modules that do not require external dependencies. This allows for more flexibility and better module support.
- The `docker_container` module has deprecated the returned fact `docker_container`. The same value is available as the returned variable `container`. The returned fact will be removed in Ansible 2.12.
- The `docker_network` module has deprecated the returned fact `docker_container`. The same value is available as the returned variable `network`. The returned fact will be removed in Ansible 2.12.
- The `docker_volume` module has deprecated the returned fact `docker_container`. The same value is available as the returned variable `volume`. The returned fact will be removed in Ansible 2.12.
- The `docker_service` module was renamed to docker_compose.
- The renamed `docker_compose` module used to return one fact per service, named same as the service. A dictionary of these facts is returned as the regular return value `services`. The returned facts will be removed in Ansible 2.12.
- **The `docker_swarm_service` module no longer sets a defaults for the following options:**
  - `user`. Before, the default was `root`.
  - `update_delay`. Before, the default was `10`.
  - `update_parallelism`. Before, the default was `1`.
- `vmware_vm_facts` used to return dict of dict with virtual machine's facts. Ansible 2.8 and onwards will return list of dict with virtual machine's facts. Please see module `vmware_vm_facts` documentation for example.
- `vmware_guest_snapshot` module used to return `results`. Since Ansible 2.8 and onwards `results` is a reserved keyword, it is replaced by `snapshot_results`. Please see module `vmware_guest_snapshots` documentation for example.
- The `panos` modules have been deprecated in favor of using the Palo Alto Networks Ansible Galaxy role. Contributions to the role can be made here.
- The `ipa_user` module originally always sent `password` to FreeIPA regardless of whether the password changed. Now the module only sends `password` if `update_password` is set to `always`, which is the default.
- The `win_psexec` has deprecated the undocumented `extra_opts` module option. This will be removed in Ansible 2.10.
- The `win_nssm` module has deprecated the following options in favor of using the `win_service` module to configure the service after installing it with `win_nssm`: * `dependencies`, use `dependencies` of `win_service` instead * `start_mode`, use `start_mode` of `win_service` instead * `user`, use `username` of `win_service` instead * `password`, use `password` of `win_service` instead These options will be removed in Ansible 2.12.
- The `win_nssm` module has also deprecated the `start`, `stop`, and `restart` values of the `status` option. You should use the `win_service` module to control the running state of the service. This will be removed in Ansible 2.12.
- The `status` module option for `win_nssm` has changed its default value to `present`. Before, the default was `start`. Consequently, the service is no longer started by default after creation with `win_nssm`, and you should use the `win_service` module to start it if needed.
- The `app_parameters` module option for `win_nssm` has been deprecated; use `argument` instead. This will be removed in Ansible 2.12.
- The `app_parameters_free_form` module option for `win_nssm` has been aliased to the new `arguments` option.

- The `win_dsc` module will now validate the input options for a DSC resource. In previous versions invalid options would be ignored but are now not.
- The `openssl_pkcs12` module will now regenerate the pkcs12 file if there are differences between the file on disk and the parameters passed to the module.

Plugins

- Ansible no longer defaults to the `paramiko` connection plugin when using macOS as the control node. Ansible will now use the `ssh` connection plugin by default on a macOS control node. Since `ssh` supports connection persistence between tasks and playbook runs, it performs better than `paramiko`. If you are using password authentication, you will need to install `sshpass` when using the `ssh` connection plugin. Or you can explicitly set the connection type to `paramiko` to maintain the pre-2.8 behavior on macOS.
- Connection plugins have been standardized to allow use of `ansible_<conn-type>_user` and `ansible_<conn-type>_password` variables. Variables such as `ansible_<conn-type>_pass` and `ansible_<conn-type>_username` are treated with lower priority than the standardized names and may be deprecated in the future. In general, the `ansible_user` and `ansible_password` vars should be used unless there is a reason to use the connection-specific variables.
- The `powershell` shell plugin now uses `async_dir` to define the async path for the results file and the default has changed to `%USERPROFILE%\.ansible_async`. To control this path now, either set the `ansible_async_dir` variable or the `async_dir` value in the `powershell` section of the config ini.
- Order of enabled inventory plugins (*INVENTORY_ENABLED*) has been updated, auto is now before yaml and ini.
- The private `_options` attribute has been removed from the `CallbackBase` class of callback plugins. If you have a third-party callback plugin which needs to access the command line arguments, use code like the following instead of trying to use `self._options`:

```
from ansible import context
[...]
tags = context.CLIARGS['tags']
```

  `context.CLIARGS` is a read-only dictionary so normal dictionary retrieval methods like `CLIARGS.get('tags')` and `CLIARGS['tags']` work as expected but you won't be able to modify the cli arguments at all.
- Play recap now counts `ignored` and `rescued` tasks as well as `ok`, `changed`, `unreachable`, `failed` and `skipped` tasks, thanks to two additional stat counters in the `default` callback plugin. Tasks that fail and have `ignore_errors: yes` set are listed as `ignored`. Tasks that fail and then execute a rescue section are listed as `rescued`. Note that `rescued` tasks are no longer counted as `failed` as in Ansible 2.7 (and earlier).
- `osx_say` callback plugin was renamed into say.
- Inventory plugins now support caching through cache plugins. To start using a cache plugin with your inventory see the section on caching in the *inventory guide*. To port a custom cache plugin to be compatible with inventory see *developer guide on cache plugins*.

Porting custom scripts

Display class

As of Ansible 2.8, the `Display` class is now a "singleton". Instead of using `__main__.display` each file should import and instantiate `ansible.utils.display.Display` on its own.

OLD In Ansible 2.7 (and earlier) the following was used to access the `display` object:

```
try:
    from __main__ import display
except ImportError:
    from ansible.utils.display import Display
    display = Display()
```

NEW In Ansible 2.8 the following should be used:

```
from ansible.utils.display import Display
display = Display()
```

Networking

- The `eos_config`, `ios_config`, and `nxos_config` modules have removed the deprecated `save` and `force` parameters, use the `save_when` parameter to replicate their functionality.
- The `nxos_vrf_af` module has removed the `safi` parameter. This parameter was deprecated in Ansible 2.4 and has had no impact on the module since then.

1.3.10 Ansible 2.7 Porting Guide

This section discusses the behavioral changes between Ansible 2.6 and Ansible 2.7.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.7 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Command Line

If you specify `--tags` or `--skip-tags` multiple times on the command line, Ansible will merge the specified tags together. In previous versions of Ansible, you could set `merge_multiple_cli_tags` to `False` if you wanted to keep only the last-specified `--tags`. This config option existed for backwards compatibility. The overwriting behavior was deprecated in 2.3 and the default behavior was changed in 2.4. Ansible-2.7 removes the config option; multiple `--tags` are now always merged.

If you have a shell script that depends on setting `merge_multiple_cli_tags` to `False`, please upgrade your script so it only adds the `--tags` you actually want before upgrading to Ansible-2.7.

Python Compatibility

Ansible has dropped compatibility with Python-2.6 on the controller (The host where **`/usr/bin/ansible`** or **`/usr/bin/ansible-playbook`** is run). Modules shipped with Ansible can still be used to manage hosts which only have Python-2.6. You just need to have a host with Python-2.7 or Python-3.5 or greater to manage those hosts from.

One thing that this does affect is the ability to use **`/usr/bin/ansible-pull`** to manage a host which has Python-2.6. `ansible-pull` runs on the host being managed but it is a controller script, not a module so it will need an updated Python. Actively developed Linux distros which ship with Python-2.6 have some means to install newer Python versions (For instance, you can install Python-2.7 through an SCL on RHEL-6) but you may need to also install Python bindings for many common modules to work (For RHEL-6, for instance, selinux bindings and yum would have to be installed for the updated Python install).

The decision to drop Python-2.6 support on the controller was made because many dependent libraries are becoming unavailable there. In particular, python-cryptography is no longer available for Python-2.6 and the last release of pycrypto (the alternative to python-cryptography) has known security bugs which will never be fixed.

Playbook

Role Precedence Fix during Role Loading

Ansible 2.7 makes a small change to variable precedence when loading roles, resolving a bug, ensuring that role loading matches *variable precedence expectations*.

Before Ansible 2.7, when loading a role, the variables defined in the role's `vars/main.yml` and `defaults/main.yml` were not available when parsing the role's `tasks/main.yml` file. This prevented the role from utilizing these variables when being parsed. The problem manifested when `import_tasks` or `import_role` was used with a variable defined in the role's vars or defaults.

In Ansible 2.7, role `vars` and `defaults` are now parsed before `tasks/main.yml`. This can cause a change in behavior if the same variable is defined at the play level and the role level with different values, and used in `import_tasks` or `import_role` to define the role or file to import.

include_role and import_role variable exposure

In Ansible 2.7 a new module argument named `public` was added to the `include_role` module that dictates whether or not the role's `defaults` and `vars` will be exposed outside of the role, allowing those variables to be used by later tasks. This value defaults to `public:  False`, matching current behavior.

`import_role` does not support the `public` argument, and will unconditionally expose the role's `defaults` and `vars` to the rest of the playbook. This functionality brings `import_role` into closer alignment with roles listed within the `roles` header in a play.

There is an important difference in the way that `include_role` (dynamic) will expose the role's variables, as opposed to `import_role` (static). `import_role` is a pre-processor, and the `defaults` and `vars` are evaluated at playbook parsing, making the variables available to tasks and roles listed at any point in the play. `include_role` is a conditional task, and the `defaults` and `vars` are evaluated at execution time, making the variables available to tasks and roles listed *after* the `include_role` task.

include_tasks/import_tasks inline variables

As of Ansible 2.7, *include_tasks* and *import_tasks* can no longer accept inline variables. Instead of using inline variables, tasks should supply variables under the `vars` keyword.

OLD In Ansible 2.6 (and earlier) the following was valid syntax for specifying variables:

```
- include_tasks: include_me.yml variable=value
```

NEW In Ansible 2.7 the task should be changed to use the `vars` keyword:

```
- include_tasks: include_me.yml
  vars:
    variable: value
```

vars_prompt with unknown algorithms

vars_prompt now throws an error if the hash algorithm specified in encrypt is not supported by the controller. This increases the safety of vars_prompt as it previously returned None if the algorithm was unknown. Some modules, notably the user module, treated a password of None as a request not to set a password. If your playbook starts erroring because of this, change the hashing algorithm being used with this filter.

Deprecated

Expedited Deprecation: Use of `__file__` in `AnsibleModule`

Note: The use of the `__file__` variable is deprecated in Ansible 2.7 and **will be eliminated in Ansible 2.8**. This is much quicker than our usual 4-release deprecation cycle.

We are deprecating the use of the `__file__` variable to refer to the file containing the currently-running code. This common Python technique for finding a filesystem path does not always work (even in vanilla Python). Sometimes a Python module can be imported from a virtual location (like inside of a zip file). When this happens, the `__file__` variable will reference a virtual location pointing to inside of the zip file. This can cause problems if, for instance, the code was trying to use `__file__` to find the directory containing the python module to write some temporary information.

Before the introduction of AnsiBallZ in Ansible 2.1, using `__file__` worked in `AnsibleModule` sometimes, but any module that used it would fail when pipelining was turned on (because the module would be piped into the python interpreter's standard input, so `__file__` wouldn't contain a file path). AnsiBallZ unintentionally made using `__file__` work, by always creating a temporary file for `AnsibleModule` to reside in.

Ansible 2.8 will no longer create a temporary file for `AnsibleModule`; instead it will read the file out of a zip file. This change should speed up module execution, but it does mean that starting with Ansible 2.8, referencing `__file__` will always fail in `AnsibleModule`.

If you are the author of a third-party module which uses `__file__` with `AnsibleModule`, please update your module(s) now, while the use of `__file__` is deprecated but still available. The most common use of `__file__` is to find a directory to write a temporary file. In Ansible 2.5 and above, you can use the `tmpdir` attribute on an `AnsibleModule` instance instead, as shown in this code from the apt module:

```diff
-    tempdir = os.path.dirname(__file__)
-    package = os.path.join(tempdir, to_native(deb.rsplit('/', 1)[1]))
+    package = os.path.join(module.tmpdir, to_native(deb.rsplit('/', 1)[1]))
```

Using a loop on a package module through squash_actions

The use of `squash_actions` to invoke a package module, such as "yum", to only invoke the module once is deprecated, and will be removed in Ansible 2.11.

Instead of relying on implicit squashing, tasks should instead supply the list directly to the `name`, `pkg` or `package` parameter of the module. This functionality has been supported in most modules since Ansible 2.3.

OLD In Ansible 2.6 (and earlier) the following task would invoke the "yum" module only 1 time to install multiple packages

```
- name: Install packages
  yum:
    name: "{{ item }}"
    state: present
  with_items: "{{ packages }}"
```

NEW In Ansible 2.7 it should be changed to look like this:

```
- name: Install packages
  yum:
    name: "{{ packages }}"
    state: present
```

Modules

Major changes in popular modules are detailed here

- The *DEFAULT_SYSLOG_FACILITY* configuration option tells Ansible modules to use a specific syslog facility when logging information on all managed machines. Due to a bug with older Ansible versions, this setting did not affect machines using journald with the systemd Python bindings installed. On those machines, Ansible log messages were sent to `/var/log/messages`, even if you set *DEFAULT_SYSLOG_FACILITY*. Ansible 2.7 fixes this bug, routing all Ansible log messages according to the value set for *DEFAULT_SYSLOG_FACILITY*. If you have *DEFAULT_SYSLOG_FACILITY* configured, the location of remote logs on systems which use journald may change.

Modules removed

The following modules no longer exist:

Deprecation notices

The following modules will be removed in Ansible 2.11. Please update your playbooks accordingly.

- `na_cdot_aggregate` use na_ontap_aggregate instead.
- `na_cdot_license` use na_ontap_license instead.
- `na_cdot_lun` use na_ontap_lun instead.
- `na_cdot_qtree` use na_ontap_qtree instead.
- `na_cdot_svm` use na_ontap_svm instead.
- `na_cdot_user` use na_ontap_user instead.
- `na_cdot_user_role` use na_ontap_user_role instead.
- `na_cdot_volume` use na_ontap_volume instead.
- `sf_account_manager` use na_elementsw_account instead.
- `sf_check_connections` use na_elementsw_check_connections instead.
- `sf_snapshot_schedule_manager` use na_elementsw_snapshot_schedule instead.
- `sf_volume_access_group_manager` use na_elementsw_access_group instead.

- `sf_volume_manager` use na_elementsw_volume instead.

Noteworthy module changes

- Check mode is now supported in the `command` and `shell` modules. However, only when `creates` or `removes` is specified. If either of these are specified, the module will check for existence of the file and report the correct changed status, if they are not included the module will skip like it had done previously.
- The `win_chocolatey` module originally required the `proxy_username` and `proxy_password` to escape any double quotes in the value. This is no longer required and the escaping may cause further issues.
- The `win_uri` module has removed the deprecated option `use_basic_parsing`, since Ansible 2.5 this option did nothing
- The `win_scheduled_task` module has removed the following deprecated options:
  - `executable`, use `path` in an actions entry instead
  - `argument`, use `arguments` in an actions entry instead
  - `store_password`, set `logon_type: password` instead
  - `days_of_week`, use `monthlydow` in a triggers entry instead
  - `frequency`, use `type`, in a triggers entry instead
  - `time`, use `start_boundary` in a triggers entry instead
- The `interface_name` module option for `na_ontap_net_vlan` has been removed and should be removed from your playbooks
- The `win_disk_image` module has deprecated the return value `mount_path`, use `mount_paths[0]` instead. This will be removed in Ansible 2.11.
- `include_role` and `include_tasks` can now be used directly from `ansible` (adhoc) and `ansible-console`:

```
#> ansible -m include_role -a 'name=myrole' all
```

- The `pip` module has added a dependency on `setuptools` to support version requirements, this requirement is for the Python interpreter that executes the module and not the Python interpreter that the module is managing.
- Prior to Ansible 2.7.10, the `replace` module did the opposite of what was intended when using the `before` and `after` options together. This now works properly but may require changes to tasks.

Plugins

- The hash_password filter now throws an error if the hash algorithm specified is not supported by the controller. This increases the safety of the filter as it previously returned None if the algorithm was unknown. Some modules, notably the user module, treated a password of None as a request not to set a password. If your playbook starts erroring because of this, change the hashing algorithm being used with this filter.

Porting custom scripts

No notable changes.

Networking

No notable changes.

1.3.11 Ansible 2.6 Porting Guide

This section discusses the behavioral changes between Ansible 2.5 and Ansible 2.6.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.6 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Playbook

- The deprecated task option `always_run` has been removed, please use `check_mode: no` instead.

Deprecated

- In the nxos_igmp_interface module, `oif_prefix` and `oif_source` properties are deprecated. Use `ois_ps` parameter with a dictionary of prefix and source to values instead.

Modules

Major changes in popular modules are detailed here:

Modules removed

The following modules no longer exist:

Deprecation notices

The following modules will be removed in Ansible 2.10. Please update your playbooks accordingly.

- `k8s_raw` use k8s instead.
- `openshift_raw` use k8s instead.
- `openshift_scale` use k8s_scale instead.

Noteworthy module changes

- The `upgrade` module option for `win_chocolatey` has been removed; use `state: latest` instead.
- The `reboot` module option for `win_feature` has been removed; use the `win_reboot` action plugin instead.
- The `win_iis_webapppool` module no longer accepts a string for the `attributes` module option; use the free form dictionary value instead.
- The `name` module option for `win_package` has been removed; this is not used anywhere and should just be removed from your playbooks.
- The `win_regedit` module no longer automatically corrects the hive path `HCCC` to `HKCC`; use `HKCC` because this is the correct hive path.
- The file_module now emits a deprecation warning when `src` is specified with a state other than `hard` or `link` as it is only supposed to be useful with those. This could have an effect on people who were depending on a buggy interaction between src and other state's to place files into a subdirectory. For instance:

  ```
  $ ansible localhost -m file -a 'path=/var/lib src=/tmp/ state=directory'
  ```

  Would create a directory named `/tmp/lib`. Instead of the above, simply spell out the entire destination path like this:

  ```
  $ ansible localhost -m file -a 'path=/tmp/lib state=directory'
  ```

- The `k8s_raw` and `openshift_raw` modules have been aliased to the new `k8s` module.
- The `k8s` module supports all Kubernetes resources including those from Custom Resource Definitions and aggregated API servers. This includes all OpenShift resources.
- The `k8s` module will not accept resources where subkeys have been snake_cased. This was a workaround that was suggested with the `k8s_raw` and `openshift_raw` modules.

- The `k8s` module may not accept resources where the `api_version` has been changed to match the shortened version in the Kubernetes Python client. You should now specify the proper full Kubernetes `api_version` for a resource.
- The `k8s` module can now process multi-document YAML files if they are passed with the `src` parameter. It will process each document as a separate resource. Resources provided inline with the `resource_definition` parameter must still be a single document.
- The `k8s` module will not automatically change `Project` creation requests into `ProjectRequest` creation requests as the `openshift_raw` module did. You must now specify the `ProjectRequest` kind explicitly.
- The `k8s` module will not automatically remove secrets from the Ansible return values (and by extension the log). In order to prevent secret values in a task from being logged, specify the `no_log` parameter on the task block.
- The `k8s_scale` module now supports scalable OpenShift objects, such as `DeploymentConfig`.
- The `lineinfile` module was changed to show a warning when using an empty string as a regexp. Since an empty regexp matches every line in a file, it will replace the last line in a file rather than inserting. If this is the desired behavior, use `'^'` which will match every line and will not trigger the warning.
- Openstack modules are no longer using `shade` library. Instead `openstacksdk` is used. Since `openstacksdk` should be already present as a dependency to `shade` no additional actions are required.

Plugins

Deprecation notices

The following modules will be removed in Ansible 2.10. Please update your playbooks accordingly.

- `openshift` use `k8s` instead.

Noteworthy plugin changes

- The `k8s` lookup plugin now supports all Kubernetes resources including those from Custom Resource Definitions and aggregated API servers. This includes all OpenShift resources.
- The `k8s` lookup plugin may not accept resources where the `api_version` has been changed to match the shortened version in the Kubernetes Python client. You should now specify the proper full Kubernetes `api_version` for a resource.
- The `k8s` lookup plugin will no longer remove secrets from the Ansible return values (and by extension the log). In order to prevent secret values in a task from being logged, specify the `no_log` parameter on the task block.

Porting custom scripts

No notable changes.

Networking

No notable changes.

Dynamic inventory scripts

- `contrib/inventory/openstack.py` has been renamed to `contrib/inventory/openstack_inventory.py`. If you have used `openstack.py` as a name for your OpenStack dynamic inventory file, change it to `openstack_inventory.py`. Otherwise the file name will conflict with imports from `openstacksdk`.

1.3.12 Ansible 2.5 Porting Guide

This section discusses the behavioral changes between Ansible 2.4 and Ansible 2.5.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.5 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Playbook

Dynamic includes and attribute inheritance

In Ansible version 2.4, the concept of dynamic includes (`include_tasks`), as opposed to static imports (`import_tasks`), was introduced to clearly define the differences in how `include` works between dynamic and static includes.

All attributes applied to a dynamic `include_*` would only apply to the include itself, while attributes applied to a static `import_*` would be inherited by the tasks within.

This separation was only partially implemented in Ansible version 2.4. As of Ansible version 2.5, this work is complete and the separation now behaves as designed; attributes applied to an `include_*` task will not be inherited by the tasks within.

To achieve an outcome similar to how Ansible worked prior to version 2.5, playbooks should use an explicit application of the attribute on the needed tasks, or use blocks to apply the attribute to many tasks. Another option is to use a static `import_*` when possible instead of a dynamic task.

OLD In Ansible 2.4:

```
- include_tasks: "{{ ansible_distribution }}.yml"
  tags:
    - distro_include
```

Included file:

```
- block:
    - debug:
        msg: "In included file"

    - apt:
```

(continues on next page)

(continued from previous page)

```
        name: nginx
        state: latest
```

NEW In Ansible 2.5:

Including task:

```
- include_tasks: "{{ ansible_distribution }}.yml"
  tags:
    - distro_include
```

Included file:

```
- block:
    - debug:
        msg: "In included file"

    - apt:
        name: nginx
        state: latest
  tags:
    - distro_include
```

The relevant change in those examples is, that in Ansible 2.5, the included file defines the tag `distro_include` again. The tag is not inherited automatically.

Fixed handling of keywords and inline variables

We made several fixes to how we handle keywords and 'inline variables', to avoid conflating the two. Unfortunately these changes mean you must specify whether *name* is a keyword or a variable when calling roles. If you have playbooks that look like this:

```
roles:
    - { role: myrole, name: Justin, othervar: othervalue, become: True}
```

You will run into errors because Ansible reads name in this context as a keyword. Beginning in 2.5, if you want to use a variable name that is also a keyword, you must explicitly declare it as a variable for the role:

```
roles:
    - { role: myrole, vars: {name: Justin, othervar: othervalue}, become: True}
```

For a full list of keywords see *Playbook Keywords*.

Migrating from with_X to loop

In most cases, loops work best with the loop keyword instead of with_X style loops. The loop syntax is usually best expressed using filters instead of more complex use of query or lookup.

These examples show how to convert many common with_ style loops to loop and filters.

with_list

with_list is directly replaced by loop.

```
- name: with_list
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_list:
    - one
    - two

- name: with_list -> loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop:
    - one
    - two
```

with_items

with_items is replaced by loop and the flatten filter.

```
- name: with_items
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_items: "{{ items }}"

- name: with_items -> loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
```

with_indexed_items

with_indexed_items is replaced by loop, the flatten filter and loop_control.index_var.

```
- name: with_indexed_items
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_indexed_items: "{{ items }}"

- name: with_indexed_items -> loop
  ansible.builtin.debug:
```

(continues on next page)

(continued from previous page)

```
    msg: "{{ index }} - {{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
  loop_control:
    index_var: index
```

with_flattened

`with_flattened` is replaced by `loop` and the `flatten` filter.

```
- name: with_flattened
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_flattened: "{{ items }}"

- name: with_flattened -> loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten }}"
```

with_together

`with_together` is replaced by `loop` and the `zip` filter.

```
- name: with_together
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_together:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_together -> loop
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ list_one|zip(list_two)|list }}"
```

Another example with complex data

```
- name: with_together -> loop
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }} - {{ item.2 }}"
  loop: "{{ data[0]|zip(*data[1:])|list }}"
  vars:
    data:
      - ['a', 'b', 'c']
      - ['d', 'e', 'f']
      - ['g', 'h', 'i']
```

with_dict

with_dict can be substituted by loop and either the dictsort or dict2items filters.

```
- name: with_dict
  ansible.builtin.debug:
    msg: "{{ item.key }} - {{ item.value }}"
  with_dict: "{{ dictionary }}"

- name: with_dict -> loop (option 1)
  ansible.builtin.debug:
    msg: "{{ item.key }} - {{ item.value }}"
  loop: "{{ dictionary|dict2items }}"

- name: with_dict -> loop (option 2)
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ dictionary|dictsort }}"
```

with_sequence

with_sequence is replaced by loop and the range function, and potentially the format filter.

```
- name: with_sequence
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_sequence: start=0 end=4 stride=2 format=testuser%02x

- name: with_sequence -> loop
  ansible.builtin.debug:
    msg: "{{ 'testuser%02x' | format(item) }}"
  loop: "{{ range(0, 4 + 1, 2)|list }}"
```

The range of the loop is exclusive of the end point.

with_subelements

with_subelements is replaced by loop and the subelements filter.

```
- name: with_subelements
  ansible.builtin.debug:
    msg: "{{ item.0.name }} - {{ item.1 }}"
  with_subelements:
    - "{{ users }}"
    - mysql.hosts

- name: with_subelements -> loop
  ansible.builtin.debug:
    msg: "{{ item.0.name }} - {{ item.1 }}"
  loop: "{{ users|subelements('mysql.hosts') }}"
```

with_nested/with_cartesian

`with_nested` and `with_cartesian` are replaced by loop and the `product` filter.

```
- name: with_nested
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_nested:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_nested -> loop
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ list_one|product(list_two)|list }}"
```

with_random_choice

`with_random_choice` is replaced by just use of the `random` filter, without need of `loop`.

```
- name: with_random_choice
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_random_choice: "{{ my_list }}"

- name: with_random_choice -> loop (No loop is needed here)
  ansible.builtin.debug:
    msg: "{{ my_list|random }}"
  tags: random
```

Deprecated

Jinja tests used as filters

Using Ansible-provided jinja tests as filters will be removed in Ansible 2.9.

Prior to Ansible 2.5, jinja tests included within Ansible were most often used as filters. The large difference in use is that filters are referenced as `variable | filter_name` while jinja tests are referenced as `variable is test_name`.

Jinja tests are used for comparisons, while filters are used for data manipulation and have different applications in jinja. This change is to help differentiate the concepts for a better understanding of jinja, and where each can be appropriately used.

As of Ansible 2.5, using an Ansible provided jinja test with filter syntax, will display a deprecation error.

OLD In Ansible 2.4 (and earlier) the use of an Ansible included jinja test would likely look like this:

```
when:
    - result | failed
    - not result | success
```

NEW In Ansible 2.5 it should be changed to look like this:

```
when:
    - result is failed
    - results is not successful
```

In addition to the deprecation warnings, many new tests have been introduced that are aliases of the old tests. These new tests make more sense grammatically with the jinja test syntax, such as the new `successful` test which aliases `success`.

```
when: result is successful
```

See *Tests* for more information.

Additionally, a script was created to assist in the conversion for tests using filter syntax to proper jinja test syntax. This script has been used to convert all of the Ansible integration tests to the correct format. There are a few limitations documented, and all changes made by this script should be evaluated for correctness before executing the modified playbooks. The script can be found at https://github.com/ansible/ansible/blob/devel/hacking/fix_test_syntax.py.

Ansible fact namespacing

Ansible facts, which have historically been written to names like `ansible_*` in the main facts namespace, have been placed in their own new namespace, `ansible_facts.*` For example, the fact `ansible_distribution` is now best queried through the variable structure `ansible_facts.distribution`.

A new configuration variable, `inject_facts_as_vars`, has been added to ansible.cfg. Its default setting, 'True', keeps the 2.4 behavior of facts variables being set in the old `ansible_*` locations (while also writing them to the new namespace). This variable is expected to be set to 'False' in a future release. When `inject_facts_as_vars` is set to False, you must refer to ansible_facts through the new `ansible_facts.*` namespace.

Modules

Major changes in popular modules are detailed here.

github_release

In Ansible versions 2.4 and older, after creating a GitHub release using the `create_release` state, the `github_release` module reported state as `skipped`. In Ansible version 2.5 and later, after creating a GitHub release using the `create_release` state, the `github_release` module now reports state as `changed`.

Modules removed

The following modules no longer exist:

- nxos_mtu use nxos_system's `system_mtu` option or nxos_interface instead
- cl_interface_policy use nclu instead
- cl_bridge use nclu instead
- cl_img_install use nclu instead
- cl_ports use nclu instead
- cl_license use nclu instead

- cl_interface use nclu instead
- cl_bond use nclu instead
- ec2_vpc use ec2_vpc_net along with supporting modules ec2_vpc_igw, ec2_vpc_route_table, ec2_vpc_subnet, ec2_vpc_dhcp_option, ec2_vpc_nat_gateway, ec2_vpc_nacl instead.
- ec2_ami_search use ec2_ami_facts instead
- docker use docker_container and docker_image instead

Note: These modules may no longer have documentation in the current release. Please see the Ansible 2.4 module documentation if you need to know how they worked for porting your playbooks.

Deprecation notices

The following modules will be removed in Ansible 2.9. Please update your playbooks accordingly.

- Apstra's `aos_*` modules are deprecated as they do not work with AOS 2.1 or higher. See new modules at https://github.com/apstra.
- nxos_ip_interface use nxos_l3_interface instead.
- nxos_portchannel use nxos_linkagg instead.
- nxos_switchport use nxos_l2_interface instead.
- panos_security_policy use panos_security_rule instead.
- panos_nat_policy use panos_nat_rule instead.
- vsphere_guest use vmware_guest instead.

Noteworthy module changes

- The stat and win_stat modules have changed the default of the option `get_md5` from `true` to `false`.

This option will be removed starting with Ansible version 2.9. The options `get_checksum: True` and `checksum_algorithm: md5` can still be used if an MD5 checksum is desired.

- `osx_say` module was renamed into say.
- Several modules which could deal with symlinks had the default value of their `follow` option changed as part of a feature to standardize the behavior of follow:
  - The file module changed from `follow=False` to `follow=True` because its purpose is to modify the attributes of a file and most systems do not allow attributes to be applied to symlinks, only to real files.
  - The replace module had its `follow` parameter removed because it inherently modifies the content of an existing file so it makes no sense to operate on the link itself.
  - The blockinfile module had its `follow` parameter removed because it inherently modifies the content of an existing file so it makes no sense to operate on the link itself.
  - In Ansible-2.5.3, the template module became more strict about its `src` file being proper utf-8. Previously, non-utf8 contents in a template module src file would result in a mangled output file (the non-utf8 characters would be replaced with a unicode replacement character). Now, on Python2, the module will error out with the message, "Template source files must be utf-8 encoded". On Python3, the module will first attempt to pass the non-utf8 characters through verbatim and fail if that does not succeed.

Plugins

As a developer, you can now use 'doc fragments' for common configuration options on plugin types that support the new plugin configuration system.

Inventory

Inventory plugins have been fine tuned, and we have started to add some common features:

- The ability to use a cache plugin to avoid costly API/DB queries is disabled by default. If using inventory scripts, some may already support a cache, but it is outside of Ansible's knowledge and control. Moving to the internal cache will allow you to use Ansible's existing cache refresh/invalidation mechanisms.
- A new 'auto' plugin, enabled by default, that can automatically detect the correct plugin to use IF that plugin is using our 'common YAML configuration format'. The previous host_list, script, yaml and ini plugins still work as they did, the auto plugin is now the last one we attempt to use. If you had customized the enabled plugins you should revise the setting to include the new auto plugin.

Shell

Shell plugins have been migrated to the new plugin configuration framework. It is now possible to customize more settings, and settings which were previously 'global' can now also be overridden using host specific variables.

For example, `system_temps` is a new setting that allows you to control what Ansible will consider a 'system temporary dir'. This is used when escalating privileges for a non-administrative user. Previously this was hardcoded to '/tmp', which some systems cannot use for privilege escalation. This setting now defaults to `[ '/var/tmp', '/tmp']`.

Another new setting is `admin_users` which allows you to specify a list of users to be considered 'administrators'. Previously this was hardcoded to `root`. It now it defaults to `[root, toor, admin]`. This information is used when choosing between your `remote_temp` and `system_temps` directory.

For a full list, check the shell plugin you are using, the default shell plugin is `sh`.

Those that had to work around the global configuration limitations can now migrate to a per host/group settings, but also note that the new defaults might conflict with existing usage if the assumptions don't correlate to your environment.

Filter

The lookup plugin API now throws an error if a non-iterable value is returned from a plugin. Previously, numbers or other non-iterable types returned by a plugin were accepted without error or warning. This change was made because plugins should always return a list. Please note that plugins that return strings and other non-list iterable values will not throw an error, but may cause unpredictable behavior. If you have a custom lookup plugin that does not return a list, you should modify it to wrap the return values in a list.

Lookup

A new option was added to lookup plugins globally named `error` which allows you to control how errors produced by the lookup are handled, before this option they were always fatal. Valid values for this option are `warn`, `ignore` and `strict`. See the *lookup* page for more details.

Porting custom scripts

No notable changes.

Network

Expanding documentation

We're expanding the network documentation. There's new content and a new Ansible Network landing page. We will continue to build the network-related documentation moving forward.

Top-level connection arguments will be removed in 2.9

Top-level connection arguments like `username`, `host`, and `password` are deprecated and will be removed in version 2.9.

OLD In Ansible < 2.4

```
- name: example of using top-level options for connection properties
  ios_command:
    commands: show version
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    authorize: yes
    auth_pass: cisco
```

The deprecation warnings reflect this schedule. The task above, run in Ansible 2.5, will result in:

```
[DEPRECATION WARNING]: Param 'username' is deprecated. See the module docs for more␣
↪information. This feature will be removed in version
2.9. Deprecation warnings can be disabled by setting deprecation_warnings=False in␣
↪ansible.cfg.
[DEPRECATION WARNING]: Param 'password' is deprecated. See the module docs for more␣
↪information. This feature will be removed in version
2.9. Deprecation warnings can be disabled by setting deprecation_warnings=False in␣
↪ansible.cfg.
[DEPRECATION WARNING]: Param 'host' is deprecated. See the module docs for more␣
↪information. This feature will be removed in version 2.9.
Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.
↪cfg.
```

We recommend using the new connection types `network_cli` and `netconf` (see below), using standard Ansible connection properties, and setting those properties in inventory by group. As you update your playbooks and inventory files, you can easily make the change to `become` for privilege escalation (on platforms that support it). For more information, see the *using become with network modules* guide and the *platform documentation*.

Adding persistent connection types `network_cli` and `netconf`

Ansible 2.5 introduces two top-level persistent connection types, `network_cli` and `netconf`. With `connection: local`, each task passed the connection parameters, which had to be stored in your playbooks. With `network_cli` and `netconf` the playbook passes the connection parameters once, so you can pass them at the command line if you prefer. We recommend you use `network_cli` and `netconf` whenever possible. Note that eAPI and NX-API still require `local` connections with `provider` dictionaries. See the *platform documentation* for more information. Unless you need a `local` connection, update your playbooks to use `network_cli` or `netconf` and to specify your connection variables with standard Ansible connection variables:

OLD In Ansible 2.4

```
---
vars:
    cli:
       host: "{{ inventory_hostname }}"
       username: operator
       password: secret
       transport: cli

tasks:
- nxos_config:
    src: config.j2
    provider: "{{ cli }}"
    username: admin
    password: admin
```

NEW In Ansible 2.5

```
[nxos:vars]
ansible_connection=network_cli
ansible_network_os=nxos
ansible_user=operator
ansible_password=secret
```

```
tasks:
- nxos_config:
    src: config.j2
```

Using a provider dictionary with either `network_cli` or `netconf` will result in a warning.

Developers: Shared Module Utilities Moved

Beginning with Ansible 2.5, shared module utilities for network modules moved to `ansible.module_utils.network`.

- Platform-independent utilities are found in `ansible.module_utils.network.common`
- Platform-specific utilities are found in `ansible.module_utils.network.{{ platform }}`

If your module uses shared module utilities, you must update all references. For example, change:

OLD In Ansible 2.4

```
from ansible.module_utils.vyos import get_config, load_config
```

NEW In Ansible 2.5

```
from ansible.module_utils.network.vyos.vyos import get_config, load_config
```

See the module utilities developer guide see *Using and developing module utilities* for more information.

1.3.13 Ansible 2.4 Porting Guide

This section discusses the behavioral changes between Ansible 2.3 and Ansible 2.4.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.4 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Python version

Ansible will not support Python 2.4 or 2.5 on the target hosts anymore. Going forward, Python 2.6+ will be required on targets, as already is the case on the controller.

Inventory

Inventory has been refactored to be implemented through plugins and now allows for multiple sources. This change is mostly transparent to users.

One exception is the `inventory_dir`, which is now a host variable; previously it could only have one value so it was set globally. This means you can no longer use it early in plays to determine `hosts:` or similar keywords. This also changes the behaviour of `add_hosts` and the implicit localhost; because they no longer automatically inherit the global value, they default to `None`. See the module documentation for more information.

The `inventory_file` remains mostly unchanged, as it was always host specific.

Since there is no longer a single inventory, the 'implicit localhost' doesn't get either of these variables defined.

A bug was fixed with the inventory path/directory, which was defaulting to the current working directory. This caused `group_vars` and `host_vars` to be picked up from the current working directory instead of just adjacent to the playbook or inventory directory when a host list (comma separated host names) was provided as inventory.

Initial playbook relative group_vars and host_vars

In Ansible versions prior to 2.4, the inventory system would maintain the context of the initial playbook that was executed. This allowed successively included playbooks from other directories to inherit group_vars and host_vars placed relative to the top level playbook file.

Due to some behavioral inconsistencies, this functionality will not be included in the new inventory system starting with Ansible version 2.4.

Similar functionality can still be achieved by using vars_files, include_vars, or group_vars and host_vars placed relative to the inventory file.

Deprecated

Specifying Inventory sources

Use of `--inventory-file` on the command line is now deprecated. Use `--inventory` or `-i`. The associated ini configuration key, `hostfile`, and environment variable, `ANSIBLE_HOSTS`, are also deprecated. Replace them with the configuration key `inventory` and environment variable *`ANSIBLE_INVENTORY`*.

Use of multiple tags

Specifying `--tags` (or `--skip-tags`) multiple times on the command line currently leads to the last one overriding all the previous ones. This behavior is deprecated. In the future, if you specify –tags multiple times the tags will be merged together. From now on, using `--tags` multiple times on one command line will emit a deprecation warning. Setting the `merge_multiple_cli_tags` option to True in the `ansible.cfg` file will enable the new behavior.

In 2.4, the default has change to merge the tags. You can enable the old overwriting behavior through the config option.

In 2.5, multiple `--tags` options will be merged with no way to go back to the old behavior.

Other caveats

No major changes in this version.

Modules

Major changes in popular modules are detailed here

- The win_shell and win_command modules now properly preserve quoted arguments in the command-line. Tasks that attempted to work around the issue by adding extra quotes/escaping may need to be reworked to remove the superfluous escaping. See Issue 23019 for additional detail.

Modules removed

The following modules no longer exist:

- None

Deprecation notices

The following modules will be removed in Ansible 2.8. Please update your playbooks accordingly.

- azure, use azure_rm_virtualmachine, which uses the new Resource Manager SDK.
- win_msi, use win_package instead

Noteworthy module changes

- The win_get_url module has the dictionary `win_get_url` in its results deprecated, its content is now also available directly in the resulting output, like other modules. This dictionary will be removed in Ansible 2.8.
- The win_unzip module no longer includes the dictionary `win_unzip` in its results; the contents are now included directly in the resulting output, like other modules.
- The win_package module return values `exit_code` and `restart_required` have been deprecated in favor of `rc` and `reboot_required` respectively. The deprecated return values will be removed in Ansible 2.6.

Plugins

A new way to configure and document plugins has been introduced. This does not require changes to existing setups but developers should start adapting to the new infrastructure now. More details will be available in the developer documentation for each plugin type.

Vars plugin changes

There have been many changes to the implementation of vars plugins, but both users and developers should not need to change anything to keep current setups working. Developers should consider changing their plugins take advantage of new features.

The most notable difference to users is that vars plugins now get invoked on demand instead of at inventory build time. This should make them more efficient for large inventories, especially when using a subset of the hosts.

Note:

- This also creates a difference with group/host_vars when using them adjacent to playbooks. Before, the 'first' playbook loaded determined the variables; now the 'current' playbook does. We are looking to fix this soon, since 'all playbooks' in the path should be considered for variable loading.
- In 2.4.1 we added a toggle to allow you to control this behaviour, 'top' will be the pre 2.4, 'bottom' will use the current playbook hosting the task and 'all' will use them all from top to bottom.

Inventory plugins

Developers should start migrating from hardcoded inventory with dynamic inventory scripts to the new Inventory Plugins. The scripts will still work through the `script` inventory plugin but Ansible development efforts will now concentrate on writing plugins rather than enhancing existing scripts.

Both users and developers should look into the new plugins because they are intended to alleviate the need for many of the hacks and workarounds found in the dynamic inventory scripts.

Callback plugins

Users:

- Callbacks are now using the new configuration system. Users should not need to change anything as the old system still works, but you might see a deprecation notice if any callbacks used are not inheriting from the built in classes. Developers need to update them as stated below.

Developers:

- If your callback does not inherit from `CallbackBase` (directly or indirectly through another callback), it will still work, but issue a deprecation notice. To avoid this and ensure it works in the future change it to inherit from `CallbackBase` so it has the new options handling methods and properties. You can also implement the new options handling methods and properties but that won't automatically inherit changes added in the future. You can look at `CallbackBase` itself and/or `AnsiblePlugin` for details.
- Any callbacks inheriting from other callbacks might need to also be updated to contain the same documented options as the parent or the options won't be available. This is noted in the developer guide.

Template lookup plugin: Escaping Strings

Prior to Ansible 2.4, backslashes in strings passed to the template lookup plugin would be escaped automatically. In 2.4, users are responsible for escaping backslashes themselves. This change brings the template lookup plugin inline with the template module so that the same backslash escaping rules apply to both.

If you have a template lookup like this:

```
- debug:
    msg: '{{ lookup("template", "template.j2") }}'
```

OLD In Ansible 2.3 (and earlier) `template.j2` would look like this:

```
{{ "name surname" | regex_replace("^[^\s]+\s+(.*)", "\1") }}
```

NEW In Ansible 2.4 it should be changed to look like this:

```
{{ "name surname" | regex_replace("^[^\\s]+\\s+(.*)", "\\1") }}
```

Tests

Tests succeeded/failed

Prior to Ansible version 2.4, a task return code of `rc` would override a return code of `failed`. In version 2.4, both `rc` and `failed` are used to calculate the state of the task. Because of this, test plugins `succeeded/failed`` have also been changed. This means that overriding a task failure with `failed_when:  no` will result in `succeeded/failed` returning `True/False`. For example:

```
- command: /bin/false
  register: result
  failed_when: no

- debug:
    msg: 'This is printed on 2.3'
  when: result|failed

- debug:
    msg: 'This is printed on 2.4'
  when: result|succeeded

- debug:
    msg: 'This is always printed'
  when: result.rc != 0
```

As we can see from the example above, in Ansible 2.3 `succeeded/failed` only checked the value of `rc`.

Networking

There have been a number of changes to how Networking Modules operate.

Playbooks should still use `connection: local`.

Persistent Connection

The configuration variables `connection_retries` and `connect_interval` which were added in Ansible 2.3 are now deprecated. For Ansible 2.4 and later use `connection_retry_timeout`.

To control timeouts use `command_timeout` rather than the previous top level `timeout` variable under `[default]`

See *Ansible Network debug guide* for more information.

Configuration

The configuration system has had some major changes. Users should be unaffected except for the following:

- All relative paths defined are relative to the *ansible.cfg* file itself. Previously they varied by setting. The new behavior should be more predictable.
- A new macro `{{CWD}}` is available for paths, which will make paths relative to the 'current working directory', this is unsafe but some users really want to rely on this behaviour.

Developers that were working directly with the previous API should revisit their usage as some methods (for example, `get_config`) were kept for backwards compatibility but will warn users that the function has been deprecated.

The new configuration has been designed to minimize the need for code changes in core for new plugins. The plugins just need to document their settings and the configuration system will use the documentation to provide what they need. This is still a work in progress; currently only 'callback' and 'connection' plugins support this. More details will be added to the specific plugin developer guides.

1.3.14 Ansible 2.3 Porting Guide

This section discusses the behavioral changes between Ansible 2.2 and Ansible 2.3.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.3 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Playbook

Restructured async to work with action plugins

In Ansible 2.2 (and possibly earlier) the *async:* keyword could not be used in conjunction with the action plugins such as *service*. This limitation has been removed in Ansible 2.3

NEW In Ansible 2.3:

```
- name: Install nginx asynchronously
  service:
    name: nginx
    state: restarted
  async: 45
```

OpenBSD version facts

The *ansible_distribution_release* and *ansible_distribution_version* facts on OpenBSD hosts were reversed in Ansible 2.2 and earlier. This has been changed so that version has the numeric portion and release has the name of the release.

OLD In Ansible 2.2 (and earlier)

```
"ansible_distribution": "OpenBSD"
"ansible_distribution_release": "6.0",
"ansible_distribution_version": "release",
```

NEW In Ansible 2.3:

```
"ansible_distribution": "OpenBSD",
"ansible_distribution_release": "release",
"ansible_distribution_version": "6.0",
```

Names Blocks

Blocks can now have names, this allows you to avoid the ugly *# this block is for…* comments.

NEW In Ansible 2.3:

```
- name: Block test case
  hosts: localhost
  tasks:
   - name: Attempt to setup foo
     block:
       - debug: msg='I execute normally'
       - command: /bin/false
       - debug: msg='I never execute, cause ERROR!'
     rescue:
       - debug: msg='I caught an error'
       - command: /bin/false
       - debug: msg='I also never execute :-('
     always:
       - debug: msg="this always executes"
```

Use of multiple tags

Specifying `--tags` (or `--skip-tags`) multiple times on the command line currently leads to the last specified tag overriding all the other specified tags. This behaviour is deprecated. In the future, if you specify –tags multiple times the tags will be merged together. From now on, using `--tags` multiple times on one command line will emit a deprecation warning. Setting the `merge_multiple_cli_tags` option to True in the `ansible.cfg` file will enable the new behaviour.

In 2.4, the default will be to merge the tags. You can enable the old overwriting behavior through the config option. In 2.5, multiple `--tags` options will be merged with no way to go back to the old behaviour.

Other caveats

Here are some rare cases that might be encountered when updating. These are mostly caused by the more stringent parser validation and the capture of errors that were previously ignored.

- Made `any_errors_fatal` inheritable from play to task and all other objects in between.

Modules

No major changes in this version.

Modules removed

No major changes in this version.

Deprecation notices

The following modules will be removed in Ansible 2.5. Please update your playbooks accordingly.

- ec2_vpc
- cl_bond
- cl_bridge
- cl_img_install
- cl_interface
- cl_interface_policy
- cl_license
- cl_ports
- nxos_mtu use nxos_system instead

Note: These modules may no longer have documentation in the current release. Please see the Ansible 2.3 module documentation if you need to know how they worked for porting your playbooks.

Noteworthy module changes

AWS lambda

Previously ignored changes that only affected one parameter. Existing deployments may have outstanding changes that this bug fix will apply.

Mount

Mount: Some fixes so bind mounts are not mounted each time the playbook runs.

Plugins

No major changes in this version.

Porting custom scripts

No major changes in this version.

Networking

There have been a number of changes to number of changes to how Networking Modules operate.

Playbooks should still use `connection: local`.

The following changes apply to:

- dellos6
- dellos9
- dellos10
- eos
- ios
- iosxr
- junos
- sros
- vyos

Deprecation of top-level connection arguments

OLD In Ansible 2.2:

```
- name: example of using top-level options for connection properties
  ios_command:
    commands: show version
    host: "{{ inventory_hostname }}"
    username: cisco
    password: cisco
    authorize: yes
    auth_pass: cisco
```

Will result in:

```
[WARNING]: argument username has been deprecated and will be removed in a future version
[WARNING]: argument host has been deprecated and will be removed in a future version
[WARNING]: argument password has been deprecated and will be removed in a future version
```

NEW In Ansible 2.3:

```
- name: Gather facts
  eos_facts:
    gather_subset: all
    provider:
      username: myuser
      password: "{{ networkpassword }}"
```

(continues on next page)

(continued from previous page)

```
      transport: cli
      host: "{{ ansible_host }}"
```

ProxyCommand replaces delegate_to

The new connection framework for Network Modules in Ansible 2.3 that uses `cli` transport no longer supports the use of the `delegate_to` directive. In order to use a bastion or intermediate jump host to connect to network devices over `cli` transport, network modules now support the use of `ProxyCommand`.

To use `ProxyCommand` configure the proxy settings in the Ansible inventory file to specify the proxy host through `ansible_ssh_common_args`.

For details on how to do this see the *network proxy guide*.

1.3.15 Ansible 2.0 Porting Guide

This section discusses the behavioral changes between Ansible 1.x and Ansible 2.0.

It is intended to assist in updating your playbooks, plugins and other parts of your Ansible infrastructure so they will work with this version of Ansible.

We suggest you read this page along with Ansible Changelog for 2.0 to understand what updates you may need to make.

This document is part of a collection on porting. The complete list of porting guides can be found at *porting guides*.

Topics

Playbook

This section discusses any changes you may need to make to your playbooks.

- Syntax in 1.9.x

```
- debug:
    msg: "{{ 'test1_junk 1\\\\3' | regex_replace('(.*)_junk (.*)', '\\\\1 \\\\2') }}"
```

- Syntax in 2.0.x

```
- debug:
    msg: "{{ 'test1_junk 1\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\2') }}"
```

- Output:

```
"msg": "test1 1\\3"
```

To make an escaped string that will work on all versions you have two options:

```
- debug: msg="{{ 'test1_junk 1\\3' | regex_replace('(.*)_junk (.*)', '\\1 \\2') }}"
```

uses key=value escaping which has not changed. The other option is to check for the ansible version:

```
"{{ (ansible_version|version_compare('2.0', 'ge'))|ternary( 'test1_junk 1\\3' | regex_
↪replace('(.*)_junk (.*)', '\\1 \\2') , 'test1_junk 1\\\\3' | regex_replace('(.*)_junk␣
↪(.*)', '\\\\1 \\\\2') ) }}"
```

- trailing newline When a string with a trailing newline was specified in the playbook through yaml dict format, the trailing newline was stripped. When specified in key=value format, the trailing newlines were kept. In v2, both methods of specifying the string will keep the trailing newlines. If you relied on the trailing newline being stripped, you can change your playbook using the following as an example:

  ```
  * Syntax in 1.9.x
  ```

  ```
  vars:
    message: >
      Testing
      some things
  tasks:
  - debug:
      msg: "{{ message }}"
  ```

  – Syntax in 2.0.x

  ```
  vars:
    old_message: >
      Testing
      some things
    message: "{{ old_message[:-1] }}"
  - debug:
      msg: "{{ message }}"
  ```

  – Output

```
"msg": "Testing some things"
```

- Behavior of templating DOS-type text files changes with Ansible v2.

  A bug in Ansible v1 causes DOS-type text files (using a carriage return and newline) to be templated to Unix-type text files (using only a newline). In Ansible v2 this long-standing bug was finally fixed and DOS-type text files are preserved correctly. This may be confusing when you expect your playbook to not show any differences when migrating to Ansible v2, while in fact you will see every DOS-type file being completely replaced (with what appears to be the exact same content).

- When specifying complex args as a variable, the variable must use the full jinja2 variable syntax (`` `{{var_name}}` ``) - bare variable names there are no longer accepted. In fact, even specifying args with variables has been deprecated, and will not be allowed in future versions:

```
---
- hosts: localhost
  connection: local
  gather_facts: false
  vars:
    my_dirs:
      - { path: /tmp/3a, state: directory, mode: 0755 }
      - { path: /tmp/3b, state: directory, mode: 0700 }
  tasks:
    - file:
      args: "{{item}}"
      with_items: "{{my_dirs}}"
```

- porting task includes
- More dynamic. Corner-case formats that were not supposed to work now do not, as expected.
- variables defined in the yaml dict format see issue 13324
- templating (variables in playbooks and template lookups) has improved with regard to keeping the original instead of turning everything into a string. If you need the old behavior, quote the value to pass it around as a string.
- Empty variables and variables set to null in yaml are no longer converted to empty strings. They will retain the value of *None*. You can override the *null_representation* setting to an empty string in your config file by setting the `ANSIBLE_NULL_REPRESENTATION` environment variable.
- Extras callbacks must be enabled in ansible.cfg. Copying is no longer necessary but you must enable them in ansible.cfg.
- dnf module has been rewritten. Some minor changes in behavior may be observed.
- win_updates has been rewritten and works as expected now.
- from 2.0.1 onwards, the implicit setup task from gather_facts now correctly inherits everything from play, but this might cause issues for those setting *environment* at the play level and depending on *ansible_env* existing. Previously this was ignored but now might issue an 'Undefined' error.

Deprecated

While all items listed here will show a deprecation warning message, they still work as they did in 1.9.x. Please note that they will be removed in 2.2 (Ansible always waits two major releases to remove a deprecated feature).

- Bare variables in `with_` loops should instead use the `"{{ var }}"` syntax, which helps eliminate ambiguity.
- The ansible-galaxy text format requirements file. Users should use the YAML format for requirements instead.
- Undefined variables within a `with_` loop's list currently do not interrupt the loop, but they do issue a warning; in the future, they will issue an error.
- Using dictionary variables to set all task parameters is unsafe and will be removed in a future version. Example of a deprecated variant:

```
- hosts: localhost
  gather_facts: no
  vars:
    debug_params:
      msg: "hello there"
  tasks:
    - debug: "{{debug_params}}"
    - debug:
      args: "{{debug_params}}"
```

Example of a recommended variant:

```
- hosts: localhost
  gather_facts: no
  vars:
    debug_params:
      msg: "hello there"
  tasks:
    - debug:
      msg: "{{debug_params['msg']}}"
```

- Host patterns should use a comma (,) or colon (:) instead of a semicolon (;) to separate hosts/groups in the pattern.
- Ranges specified in host patterns should use the [x:y] syntax, instead of [x-y].
- Playbooks using privilege escalation should always use “become*” options rather than the old su*/sudo* options.
- The “short form” for vars_prompt is no longer supported. For example:

  ```
  vars_prompt:
      variable_name: "Prompt string"
  ```

- Specifying variables at the top level of a task include statement is no longer supported. For example:

  ```
  - include_tasks: foo.yml
      a: 1
  ```

Should now be:

```
- include_tasks: foo.yml
  vars:
    a: 1
```

- Setting any_errors_fatal on a task is no longer supported. This should be set at the play level only.
- Bare variables in the *environment* dictionary (for plays/tasks/and so on) are no longer supported. Variables specified there should use the full variable syntax: '{{foo}}'.
- Tags (or any directive) should no longer be specified with other parameters in a task include. Instead, they should be specified as an option on the task. For example:

  ```
  - include_tasks: foo.yml tags=a,b,c
  ```

  Should be:

  ```
  - include_tasks: foo.yml
    tags: [a, b, c]
  ```

- The first_available_file option on tasks has been deprecated. Users should use the with_first_found option or lookup ('first_found', . . .) plugin.

Other caveats

Here are some corner cases encountered when updating. These are mostly caused by the more stringent parser validation and the capture of errors that were previously ignored.

- Bad variable composition:

  ```
  with_items: myvar_{{rest_of_name}}
  ```

  This worked 'by accident' as the errors were retemplated and ended up resolving the variable, it was never intended as valid syntax and now properly returns an error, use the following instead.:

  ```
  hostvars[inventory_hostname]['myvar_' + rest_of_name]
  ```

- Misspelled directives:

  ```
  - task: dostuf
    becom: yes
  ```

  The task always ran without using privilege escalation (for that you need *become*) but was also silently ignored so the play 'ran' even though it should not, now this is a parsing error.

- Duplicate directives:

  ```
  - task: dostuf
    when: True
    when: False
  ```

  The first *when* was ignored and only the 2nd one was used as the play ran w/o warning it was ignoring one of the directives, now this produces a parsing error.

- Conflating variables and directives:

  ```
  - role: {name=rosy, port=435 }

  # in tasks/main.yml
  - wait_for: port={{port}}
  ```

The *port* variable is reserved as a play/task directive for overriding the connection port, in previous versions this got conflated with a variable named *port* and was usable later in the play, this created issues if a host tried to reconnect or was using a non caching connection. Now it will be correctly identified as a directive and the *port* variable will appear as undefined, this now forces the use of non conflicting names and removes ambiguity when adding settings and variables to a role invocation.

- Bare operations on *with_*:

```
with_items: var1 + var2
```

An issue with the 'bare variable' features, which was supposed only template a single variable without the need of braces ({{)}}, would in some versions of Ansible template full expressions. Now you need to use proper templating and braces for all expressions everywhere except conditionals (*when*):

```
with_items: "{{var1 + var2}}"
```

The bare feature itself is deprecated as an undefined variable is indistinguishable from a string which makes it difficult to display a proper error.

Porting plugins

In ansible-1.9.x, you would generally copy an existing plugin to create a new one. Simply implementing the methods and attributes that the caller of the plugin expected made it a plugin of that type. In ansible-2.0, most plugins are implemented by subclassing a base class for each plugin type. This way the custom plugin does not need to contain methods which are not customized.

Lookup plugins

- lookup plugins ; import version

Connection plugins

- connection plugins

Action plugins

- action plugins

Callback plugins

Although Ansible 2.0 provides a new callback API the old one continues to work for most callback plugins. However, if your callback plugin makes use of `self.playbook`, `self.play`, or `self.task` then you will have to store the values for these yourself as ansible no longer automatically populates the callback with them. Here's a short snippet that shows you how:

```
import os
from ansible.plugins.callback import CallbackBase

class CallbackModule(CallbackBase):
    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        self.playbook = None
        self.playbook_name = None
        self.play = None
        self.task = None

    def v2_playbook_on_start(self, playbook):
        self.playbook = playbook
        self.playbook_name = os.path.basename(self.playbook._file_name)

    def v2_playbook_on_play_start(self, play):
        self.play = play

    def v2_playbook_on_task_start(self, task, is_conditional):
        self.task = task

    def v2_on_any(self, *args, **kwargs):
        self._display.display('%s: %s: %s' % (self.playbook_name,
        self.play.name, self.task))
```

Connection plugins

- connection plugins

Hybrid plugins

In specific cases you may want a plugin that supports both ansible-1.9.x *and* ansible-2.0. Much like porting plugins from v1 to v2, you need to understand how plugins work in each version and support both requirements.

Since the ansible-2.0 plugin system is more advanced, it is easier to adapt your plugin to provide similar pieces (subclasses, methods) for ansible-1.9.x as ansible-2.0 expects. This way your code will look a lot cleaner.

You may find the following tips useful:

- Check whether the ansible-2.0 class(es) are available and if they are missing (ansible-1.9.x) mimic them with the needed methods (for example, `__init__`)
- When ansible-2.0 python modules are imported, and they fail (ansible-1.9.x), catch the `ImportError` exception and perform the equivalent imports for ansible-1.9.x. With possible translations (for example, importing specific methods).
- Use the existence of these methods as a qualifier to what version of Ansible you are running. So rather than using version checks, you can do capability checks instead. (See examples below)
- Document for each if-then-else case for which specific version each block is needed. This will help others to understand how they have to adapt their plugins, but it will also help you to remove the older ansible-1.9.x support when it is deprecated.
- When doing plugin development, it is very useful to have the `warning()` method during development, but it is also important to emit warnings for deadends (cases that you expect should never be triggered) or corner cases (for example, cases where you expect misconfigurations).
- It helps to look at other plugins in ansible-1.9.x and ansible-2.0 to understand how the API works and what modules, classes and methods are available.

Lookup plugins

As a simple example we are going to make a hybrid `fileglob` lookup plugin.

```python
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

import os
import glob

try:
    # ansible-2.0
    from ansible.plugins.lookup import LookupBase
except ImportError:
    # ansible-1.9.x

    class LookupBase(object):
        def __init__(self, basedir=None, runner=None, **kwargs):
            self.runner = runner
            self.basedir = self.runner.basedir

        def get_basedir(self, variables):
            return self.basedir

try:
    # ansible-1.9.x
    from ansible.utils import (listify_lookup_plugin_terms, path_dwim, warning)
except ImportError:
    # ansible-2.0
    from ansible.utils.display import Display
    warning = Display().warning

class LookupModule(LookupBase):

    # For ansible-1.9.x, we added inject=None as valid argument
    def run(self, terms, inject=None, variables=None, **kwargs):

        # ansible-2.0, but we made this work for ansible-1.9.x too !
        basedir = self.get_basedir(variables)

        # ansible-1.9.x
        if 'listify_lookup_plugin_terms' in globals():
            terms = listify_lookup_plugin_terms(terms, basedir, inject)

        ret = []
        for term in terms:
            term_file = os.path.basename(term)

            # For ansible-1.9.x, we imported path_dwim() from ansible.utils
            if 'path_dwim' in globals():
                # ansible-1.9.x
                dwimmed_path = path_dwim(basedir, os.path.dirname(term))
            else:
```

(continues on next page)

(continued from previous page)

```
                # ansible-2.0
                dwimmed_path = self._loader.path_dwim_relative(basedir, 'files', os.path.
↪dirname(term))

            globbed = glob.glob(os.path.join(dwimmed_path, term_file))
            ret.extend(g for g in globbed if os.path.isfile(g))

        return ret
```

Note: In the above example we did not use the `warning()` method as we had no direct use for it in the final version. However we left this code in so people can use this part during development/porting/use.

Connection plugins

- connection plugins

Action plugins

- action plugins

Callback plugins

- callback plugins

Connection plugins

- connection plugins

Porting custom scripts

Custom scripts that used the `ansible.runner.Runner` API in 1.x have to be ported in 2.x. Please refer to: *Python API*

1.4 Building Ansible inventories

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the guide to building Ansible inventories. An inventory is a list of managed nodes, or hosts, that Ansible deploys and configures. This guide introduces you to inventories and covers the following topics:

- Creating inventories to track list of servers and devices that you want to automate.
- Using dynamic inventories to track cloud services with servers and devices that are constantly starting and stopping.
- Using patterns to automate specific sub-sets of an inventory.
- Expanding and refining the connection methods Ansible uses for your inventory.

1.4.1 How to build your inventory

Ansible automates tasks on managed nodes or "hosts" in your infrastructure, using a list or group of lists known as inventory. You can pass host names at the command line, but most Ansible users create inventory files. Your inventory defines the managed nodes you automate, with groups so you can run automation tasks on multiple hosts at the same time. Once your inventory is defined, you use *patterns* to select the hosts or groups you want Ansible to run against.

The simplest inventory is a single file with a list of hosts and groups. The default location for this file is `/etc/ansible/hosts`. You can specify a different inventory file at the command line using the `-i <path>` option or in configuration using `inventory`.

Ansible *Inventory plugins* support a range of formats and sources to make your inventory flexible and customizable. As your inventory expands, you may need more than a single file to organize your hosts and groups. Here are three options beyond the `/etc/ansible/hosts` file: - You can create a directory with multiple inventory files. See *Organizing inventory in a directory*. These can use different formats (YAML, ini, and so on). - You can pull inventory dynamically. For example, you can use a dynamic inventory plugin to list resources in one or more cloud providers. See *Working with dynamic inventory*. - You can use multiple sources for inventory, including both dynamic inventory and static files. See *Passing multiple inventory sources*.

Inventory basics: formats, hosts, and groups

You can create your inventory file in one of many formats, depending on the inventory plugins you have. The most common formats are INI and YAML. A basic INI `/etc/ansible/hosts` might look like this:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose. Group names should follow the same guidelines as *Creating valid variable names*.

Here's that same basic inventory file in YAML format:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Default groups

Even if you do not define any groups in your inventory file, Ansible creates two default groups: `all` and `ungrouped`. The `all` group contains every host. The `ungrouped` group contains all hosts that don't have another group aside from `all`. Every host will always belong to at least 2 groups (`all` and `ungrouped` or `all` and some other group). For example, in the basic inventory above, the host `mail.example.com` belongs to the `all` group and the `ungrouped` group; the host `two.example.com` belongs to the `all` group and the `dbservers` group. Though `all` and `ungrouped` are always present, they can be implicit and not appear in group listings like `group_names`.

Hosts in multiple groups

You can (and probably will) put each host in more than one group. For example a production webserver in a datacenter in Atlanta might be included in groups called `[prod]` and `[atlanta]` and `[webservers]`. You can create groups that track:

- What - An application, stack or microservice (for example, database servers, web servers, and so on).
- Where - A datacenter or region, to talk to local DNS, storage, and so on (for example, east, west).
- When - The development stage, to avoid testing on production resources (for example, prod, test).

Extending the previous YAML inventory to include what, when, and where would look like this:

```yaml
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
    east:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    west:
      hosts:
        bar.example.com:
        three.example.com:
    prod:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    test:
      hosts:
        bar.example.com:
        three.example.com:
```

You can see that `one.example.com` exists in the `dbservers`, `east`, and `prod` groups.

Grouping groups: parent/child group relationships

You can create parent/child relationships among groups. Parent groups are also known as nested groups or groups of groups. For example, if all your production hosts are already in groups such as `atlanta_prod` and `denver_prod`, you can create a `production` group that includes those smaller groups. This approach reduces maintenance because you can add or remove hosts from the parent group by editing the child groups.

To create parent/child relationships for groups:

- in INI format, use the `:children` suffix
- in YAML format, use the `children:` entry

Here is the same inventory as shown above, simplified with parent groups for the `prod` and `test` groups. The two inventory files give you the same results:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
    east:
      hosts:
        foo.example.com:
        one.example.com:
        two.example.com:
    west:
      hosts:
        bar.example.com:
        three.example.com:
    prod:
      children:
        east:
    test:
      children:
        west:
```

Child groups have a couple of properties to note:

- Any host that is member of a child group is automatically a member of the parent group.
- Groups can have multiple parents and children, but not circular relationships.
- Hosts can also be in multiple groups, but there will only be **one** instance of a host at runtime. Ansible merges the data from the multiple groups.

Adding ranges of hosts

If you have a lot of hosts with a similar pattern, you can add them as a range rather than listing each hostname separately:

In INI:

```
[webservers]
www[01:50].example.com
```

In YAML:

```
...
  webservers:
    hosts:
      www[01:50].example.com:
```

You can specify a stride (increments between sequence numbers) when defining a numeric range of hosts:

In INI:

```
[webservers]
www[01:50:2].example.com
```

In YAML:

```
...
  webservers:
    hosts:
      www[01:50:2].example.com:
```

The example above would make the subdomains www01, www03, www05, …, www49 match, but not www00, www02, www50 and so on, because the stride (increment) is 2 units each step.

For numeric patterns, leading zeros can be included or removed, as desired. Ranges are inclusive. You can also define alphabetic ranges:

```
[databases]
db-[a:f].example.com
```

Passing multiple inventory sources

You can target multiple inventory sources (directories, dynamic inventory scripts or files supported by inventory plugins) at the same time by giving multiple inventory parameters from the command line or by configuring *ANSIBLE_INVENTORY*. This can be useful when you want to target normally separate environments, like staging and production, at the same time for a specific action.

To target two inventory sources from the command line:

```
ansible-playbook get_logs.yml -i staging -i production
```

Organizing inventory in a directory

You can consolidate multiple inventory sources in a single directory. The simplest version of this is a directory with multiple files instead of a single inventory file. A single file gets difficult to maintain when it gets too long. If you have multiple teams and multiple automation projects, having one inventory file per team or project lets everyone easily find the hosts and groups that matter to them.

You can also combine multiple inventory source types in an inventory directory. This can be useful for combining static and dynamic hosts and managing them as one inventory. The following inventory directory combines an inventory plugin source, a dynamic inventory script, and a file with static hosts:

```
inventory/
  openstack.yml          # configure inventory plugin to get hosts from OpenStack cloud
  dynamic-inventory.py   # add additional hosts with dynamic inventory script
  on-prem                # add static hosts and groups
  parent-groups          # add static hosts and groups
```

You can target this inventory directory as follows:

```
ansible-playbook example.yml -i inventory
```

You can also configure the inventory directory in your `ansible.cfg` file. See *Configuring Ansible* for more details.

Managing inventory load order

Ansible loads inventory sources in ASCII order according to the filenames. If you define parent groups in one file or directory and child groups in other files or directories, the files that define the child groups must be loaded first. If the parent groups are loaded first, you will see the error `Unable to parse /path/to/source_of_parent_groups as an inventory source`.

For example, if you have a file called `groups-of-groups` that defines a `production` group with child groups defined in a file called `on-prem`, Ansible cannot parse the `production` group. To avoid this problem, you can control the load order by adding prefixes to the files:

```
inventory/
  01-openstack.yml          # configure inventory plugin to get hosts from OpenStack␣
↪cloud
  02-dynamic-inventory.py   # add additional hosts with dynamic inventory script
  03-on-prem                # add static hosts and groups
  04-groups-of-groups       # add parent groups
```

You can find examples of how to organize your inventories and group your hosts in *Inventory setup examples*.

Adding variables to inventory

You can store variable values that relate to a specific host or group in inventory. To start with, you may add variables directly to the hosts and groups in your main inventory file.

We document adding variables in the main inventory file for simplicity. However, storing variables in separate host and group variable files is a more robust approach to describing your system policy. Setting variables in the main inventory file is only a shorthand. See *Organizing host and group variables* for guidelines on storing variable values in individual files in the 'host_vars' directory. See *Organizing host and group variables* for details.

Assigning a variable to one machine: host variables

You can easily assign a variable to a single host, then use it later in playbooks. You can do this directly in your inventory file.

In INI:

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

In YAML:

```
atlanta:
  hosts:
    host1:
      http_port: 80
      maxRequestsPerChild: 808
    host2:
      http_port: 303
      maxRequestsPerChild: 909
```

Unique values like non-standard SSH ports work well as host variables. You can add them to your Ansible inventory by adding the port number after the hostname with a colon:

```
badwolf.example.com:5309
```

Connection variables also work well as host variables:

```
[targets]

localhost              ansible_connection=local
other1.example.com     ansible_connection=ssh        ansible_user=myuser
other2.example.com     ansible_connection=ssh        ansible_user=myotheruser
```

Note: If you list non-standard SSH ports in your SSH config file, the `openssh` connection will find and use them, but the `paramiko` connection will not.

Inventory aliases

You can also define aliases in your inventory using host variables:

In INI:

```
jumper ansible_port=5555 ansible_host=192.0.2.50
```

In YAML:

```
...
  hosts:
    jumper:
      ansible_port: 5555
      ansible_host: 192.0.2.50
```

In this example, running Ansible against the host alias "jumper" will connect to 192.0.2.50 on port 5555. See *behavioral inventory parameters* to further customize the connection to hosts.

Defining variables in INI format

Values passed in the INI format using the `key=value` syntax are interpreted differently depending on where they are declared:

- When declared inline with the host, INI values are interpreted as Python literal structures (strings, numbers, tuples, lists, dicts, booleans, None). Host lines accept multiple `key=value` parameters per line. Therefore they need a way to indicate that a space is part of a value rather than a separator. Values that contain whitespace can be quoted (single or double). See the Python shlex parsing rules for details.
- When declared in a `:vars` section, INI values are interpreted as strings. For example `var=FALSE` would create a string equal to 'FALSE'. Unlike host lines, `:vars` sections accept only a single entry per line, so everything after the = must be the value for the entry.

If a variable value set in an INI inventory must be a certain type (for example, a string or a boolean value), always specify the type with a filter in your task. Do not rely on types set in INI inventories when consuming variables.

Consider using YAML format for inventory sources to avoid confusion on the actual type of a variable. The YAML inventory plugin processes variable values consistently and correctly.

Assigning a variable to many machines: group variables

If all hosts in a group share a variable value, you can apply that variable to an entire group at once.

In INI:

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

In YAML:

```yaml
atlanta:
  hosts:
    host1:
    host2:
  vars:
    ntp_server: ntp.atlanta.example.com
    proxy: proxy.atlanta.example.com
```

Group variables are a convenient way to apply variables to multiple hosts at once. Before executing, however, Ansible always flattens variables, including inventory variables, to the host level. If a host is a member of multiple groups, Ansible reads variable values from all of those groups. If you assign different values to the same variable in different groups, Ansible chooses which value to use based on internal *rules for merging*.

Inheriting variable values: group variables for groups of groups

You can apply variables to parent groups (nested groups or groups of groups) as well as to child groups. The syntax is the same: `:vars` for INI format and `vars:` for YAML format:

In INI:

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

In YAML:

```yaml
all:
  children:
    usa:
      children:
        southeast:
          children:
            atlanta:
              hosts:
                host1:
                host2:
            raleigh:
              hosts:
                host2:
                host3:
          vars:
            some_server: foo.southeast.example.com
            halon_system_timeout: 30
            self_destruct_countdown: 60
            escape_pods: 2
        northeast:
        northwest:
```

(continues on next page)

(continued from previous page)

```
        southwest:
```

A child group's variables will have higher precedence (override) a parent group's variables.

Organizing host and group variables

Although you can store variables in the main inventory file, storing separate host and group variables files may help you organize your variable values more easily. You can also use lists and hash data in host and group variables files, which you cannot do in your main inventory file.

Host and group variable files must use YAML syntax. Valid file extensions include '.yml', '.yaml', '.json', or no file extension. See *YAML Syntax* if you are new to YAML.

Ansible loads host and group variable files by searching paths relative to the inventory file or the playbook file. If your inventory file at /etc/ansible/hosts contains a host named 'foosball' that belongs to two groups, 'raleigh' and 'webservers', that host will use variables in YAML files at the following locations:

```
/etc/ansible/group_vars/raleigh # can optionally end in '.yml', '.yaml', or '.json'
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/foosball
```

For example, if you group hosts in your inventory by datacenter, and each datacenter uses its own NTP server and database server, you can create a file called /etc/ansible/group_vars/raleigh to store the variables for the raleigh group:

```
---
ntp_server: acme.example.org
database_server: storage.example.org
```

You can also create *directories* named after your groups or hosts. Ansible will read all the files in these directories in lexicographical order. An example with the 'raleigh' group:

```
/etc/ansible/group_vars/raleigh/db_settings
/etc/ansible/group_vars/raleigh/cluster_settings
```

All hosts in the 'raleigh' group will have the variables defined in these files available to them. This can be very useful to keep your variables organized when a single file gets too big, or when you want to use *Ansible Vault* on some group variables.

For ansible-playbook you can also add group_vars/ and host_vars/ directories to your playbook directory. Other Ansible commands (for example, ansible, ansible-console, and so on) will only look for group_vars/ and host_vars/ in the inventory directory. If you want other commands to load group and host variables from a playbook directory, you must provide the --playbook-dir option on the command line. If you load inventory files from both the playbook directory and the inventory directory, variables in the playbook directory will override variables set in the inventory directory.

Keeping your inventory file and variables in a git repo (or other version control) is an excellent way to track changes to your inventory and host variables.

How variables are merged

By default variables are merged/flattened to the specific host before a play is run. This keeps Ansible focused on the Host and Task, so groups don't really survive outside of inventory and host matching. By default, Ansible overwrites variables including the ones defined for a group and/or host (see *DEFAULT_HASH_BEHAVIOUR*). The order/precedence is (from lowest to highest):

- all group (because it is the 'parent' of all other groups)
- parent group
- child group
- host

By default Ansible merges groups at the same parent/child level in ASCII order, and variables from the last group loaded overwrite variables from the previous groups. For example, an a_group will be merged with b_group and b_group vars that match will overwrite the ones in a_group.

You can change this behavior by setting the group variable `ansible_group_priority` to change the merge order for groups of the same level (after the parent/child order is resolved). The larger the number, the later it will be merged, giving it higher priority. This variable defaults to 1 if not set. For example:

```
a_group:
  vars:
    testvar: a
    ansible_group_priority: 10
b_group:
  vars:
    testvar: b
```

In this example, if both groups have the same priority, the result would normally have been `testvar == b`, but since we are giving the `a_group` a higher priority the result will be `testvar == a`.

Note: `ansible_group_priority` can only be set in the inventory source and not in group_vars/, as the variable is used in the loading of group_vars.

Managing inventory variable load order

When using multiple inventory sources, keep in mind that any variable conflicts are resolved according to the rules described in *How variables are merged* and *Variable precedence: Where should I put a variable?*. You can control the merging order of variables in inventory sources to get the variable value you need.

When you pass multiple inventory sources at the command line, Ansible merges variables in the order you pass those parameters. If `[all:vars]` in staging inventory defines `myvar = 1` and production inventory defines `myvar = 2`, then:

- Pass `-i staging -i production` to run the playbook with `myvar = 2`.
- Pass `-i production -i staging` to run the playbook with `myvar = 1`.

When you put multiple inventory sources in a directory, Ansible merges them in ASCII order according to the filenames. You can control the load order by adding prefixes to the files:

```
inventory/
  01-openstack.yml          # configure inventory plugin to get hosts from Openstack␣
```

(continues on next page)

(continued from previous page)

```
↪cloud
  02-dynamic-inventory.py   # add additional hosts with dynamic inventory script
  03-static-inventory       # add static hosts
  group_vars/
    all.yml                 # assign variables to all hosts
```

If `01-openstack.yml` defines `myvar = 1` for the group `all`, `02-dynamic-inventory.py` defines `myvar = 2`, and `03-static-inventory` defines `myvar = 3`, the playbook will be run with `myvar = 3`.

For more details on inventory plugins and dynamic inventory scripts see *Inventory plugins* and *Working with dynamic inventory*.

Connecting to hosts: behavioral inventory parameters

As described above, setting the following variables control how Ansible interacts with remote hosts.

Host connection:

Note: Ansible does not expose a channel to allow communication between the user and the ssh process to accept a password manually to decrypt an ssh key when using the ssh connection plugin (which is the default). The use of `ssh-agent` is highly recommended.

ansible_connection
Connection type to the host. This can be the name of any of ansible's connection plugins. SSH protocol types are `smart`, `ssh` or `paramiko`. The default is smart. Non-SSH based types are described in the next section.

General for all connections:

ansible_host
The name of the host to connect to, if different from the alias you wish to give to it.

ansible_port
The connection port number, if not the default (22 for ssh)

ansible_user
The user name to use when connecting to the host

ansible_password
The password to use to authenticate to the host (never store this variable in plain text; always use a vault. See *Keep vaulted variables safely visible*)

Specific to the SSH connection:

ansible_ssh_private_key_file
Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.

ansible_ssh_common_args
This setting is always appended to the default command line for **`sftp`**, **`scp`**, and **`ssh`**. Useful to configure a `ProxyCommand` for a certain host (or group).

ansible_sftp_extra_args
This setting is always appended to the default **`sftp`** command line.

ansible_scp_extra_args
This setting is always appended to the default **`scp`** command line.

ansible_ssh_extra_args
This setting is always appended to the default **`ssh`** command line.

ansible_ssh_pipelining
Determines whether or not to use SSH pipelining. This can override the pipelining setting in ansible.cfg.

ansible_ssh_executable (added in version 2.2)
This setting overrides the default behavior to use the system **ssh**. This can override the ssh_executable setting in ansible.cfg.

Privilege escalation (see *Ansible Privilege Escalation* for further details):

ansible_become
Equivalent to ansible_sudo or ansible_su, allows to force privilege escalation

ansible_become_method
Allows to set privilege escalation method

ansible_become_user
Equivalent to ansible_sudo_user or ansible_su_user, allows to set the user you become through privilege escalation

ansible_become_password
Equivalent to ansible_sudo_password or ansible_su_password, allows you to set the privilege escalation password (never store this variable in plain text; always use a vault. See *Keep vaulted variables safely visible*)

ansible_become_exe
Equivalent to ansible_sudo_exe or ansible_su_exe, allows you to set the executable for the escalation method selected

ansible_become_flags
Equivalent to ansible_sudo_flags or ansible_su_flags, allows you to set the flags passed to the selected escalation method. This can be also set globally in ansible.cfg in the sudo_flags option

Remote host environment parameters:

ansible_shell_type
The shell type of the target system. You should not use this setting unless you have set the *ansible_shell_executable* to a non-Bourne (sh) compatible shell. By default commands are formatted using sh-style syntax. Setting this to csh or fish will cause commands executed on target systems to follow those shell's syntax instead.

ansible_python_interpreter
The target host python path. This is useful for systems with more than one Python or not located at **/usr/bin/python** such as *BSD, or where **/usr/bin/python** is not a 2.X series Python. We do not use the **/usr/bin/env** mechanism as that requires the remote user's path to be set right and also assumes the **python** executable is named python, where the executable might be named something like **python2.6**.

ansible_*_interpreter
Works for anything such as ruby or perl and works just like *ansible_python_interpreter*. This replaces shebang of modules which will run on that host.

New in version 2.1.

ansible_shell_executable
This sets the shell the ansible controller will use on the target machine, overrides executable in ansible.cfg which defaults to **/bin/sh**. You should really only change it if is not possible to use **/bin/sh** (in other words, if **/bin/sh** is not installed on the target machine or cannot be run from sudo.).

Examples from an Ansible-INI host file:

```
some_host         ansible_port=2222     ansible_user=manager
aws_host          ansible_ssh_private_key_file=/home/example/.ssh/aws.pem
```

(continues on next page)

(continued from previous page)

```
freebsd_host      ansible_python_interpreter=/usr/local/bin/python
ruby_module_host  ansible_ruby_interpreter=/usr/bin/ruby.1.9.3
```

Non-SSH connection types

As stated in the previous section, Ansible executes playbooks over SSH but it is not limited to this connection type. With the host specific parameter `ansible_connection=<connector>`, the connection type can be changed. The following non-SSH based connectors are available:

local

This connector can be used to deploy the playbook to the control machine itself.

docker

This connector deploys the playbook directly into Docker containers using the local Docker client. The following parameters are processed by this connector:

ansible_host
: The name of the Docker container to connect to.

ansible_user
: The user name to operate within the container. The user must exist inside the container.

ansible_become
: If set to `true` the `become_user` will be used to operate within the container.

ansible_docker_extra_args
: Could be a string with any additional arguments understood by Docker, which are not command specific. This parameter is mainly used to configure a remote Docker daemon to use.

Here is an example of how to instantly deploy to created containers:

```
- name: Create a jenkins container
  community.general.docker_container:
    docker_host: myserver.net:4243
    name: my_jenkins
    image: jenkins

- name: Add the container to inventory
  ansible.builtin.add_host:
    name: my_jenkins
    ansible_connection: docker
    ansible_docker_extra_args: "--tlsverify --tlscacert=/path/to/ca.pem --tlscert=/path/
↪to/client-cert.pem --tlskey=/path/to/client-key.pem -H=tcp://myserver.net:4243"
    ansible_user: jenkins
  changed_when: false

- name: Create a directory for ssh keys
  delegate_to: my_jenkins
  ansible.builtin.file:
    path: "/var/jenkins_home/.ssh/jupiter"
    state: directory
```

For a full list with available plugins and examples, see *Plugin list*.

Note: If you're reading the docs from the beginning, this may be the first example you've seen of an Ansible playbook. This is not an inventory file. Playbooks will be covered in great detail later in the docs.

Inventory setup examples

See also *Sample Ansible setup*, which shows inventory along with playbooks and other Ansible artifacts.

Example: One inventory per environment

If you need to manage multiple environments it's sometimes prudent to have only hosts of a single environment defined per inventory. This way, it is harder to, for instance, accidentally change the state of nodes inside the "test" environment when you actually wanted to update some "staging" servers.

For the example mentioned above you could have an `inventory_test` file:

```
[dbservers]
db01.test.example.com
db02.test.example.com

[appservers]
app01.test.example.com
app02.test.example.com
app03.test.example.com
```

That file only includes hosts that are part of the "test" environment. Define the "staging" machines in another file called `inventory_staging`:

```
[dbservers]
db01.staging.example.com
db02.staging.example.com

[appservers]
app01.staging.example.com
app02.staging.example.com
app03.staging.example.com
```

To apply a playbook called `site.yml` to all the app servers in the test environment, use the following command:

```
ansible-playbook -i inventory_test -l appservers site.yml
```

Example: Group by function

In the previous section you already saw an example for using groups in order to cluster hosts that have the same function. This allows you, for instance, to define firewall rules inside a playbook or role affecting only database servers:

```
- hosts: dbservers
  tasks:
  - name: Allow access from 10.0.0.1
    ansible.builtin.iptables:
```

(continues on next page)

(continued from previous page)

```
      chain: INPUT
      jump: ACCEPT
      source: 10.0.0.1
```

Example: Group by location

Other tasks might be focused on where a certain host is located. Let's say that `db01.test.example.com` and `app01.test.example.com` are located in DC1 while `db02.test.example.com` is in DC2:

```
[dc1]
db01.test.example.com
app01.test.example.com

[dc2]
db02.test.example.com
```

In practice, you might even end up mixing all these setups as you might need to, on one day, update all nodes in a specific data center while, on another day, update all the application servers no matter their location.

See also:

Inventory plugins
Pulling inventory from dynamic or static sources

Working with dynamic inventory
Pulling inventory from dynamic sources, such as cloud providers

Introduction to ad hoc commands
Examples of basic commands

Working with playbooks
Learning Ansible's configuration, deployment, and orchestration language.

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.4.2 Working with dynamic inventory

If your Ansible inventory fluctuates over time, with hosts spinning up and shutting down in response to business demands, the static inventory solutions described in *How to build your inventory* will not serve your needs. You may need to track hosts from multiple sources: cloud providers, LDAP, Cobbler, and/or enterprise CMDB systems.

Ansible integrates all of these options through a dynamic external inventory system. Ansible supports two ways to connect with external inventory: *Inventory plugins* and *inventory scripts*.

Inventory plugins take advantage of the most recent updates to the Ansible core code. We recommend plugins over scripts for dynamic inventory. You can *write your own plugin* to connect to additional dynamic inventory sources.

You can still use inventory scripts if you choose. When we implemented inventory plugins, we ensured backwards compatibility through the script inventory plugin. The examples below illustrate how to use inventory scripts.

If you prefer a GUI for handling dynamic inventory, the inventory database on AWX or *Red Hat Ansible Automation Platform* syncs with all your dynamic inventory sources, provides web and REST access to the results, and offers a graphical inventory editor. With a database record of all of your hosts, you can correlate past event history and see which hosts have had failures on their last playbook runs.

Inventory script example: Cobbler

Ansible integrates seamlessly with Cobbler, a Linux installation server originally written by Michael DeHaan and now led by James Cammarata, who works for Ansible.

While primarily used to kickoff OS installations and manage DHCP and DNS, Cobbler has a generic layer that can represent data for multiple configuration management systems (even at the same time) and serve as a 'lightweight CMDB'.

To tie your Ansible inventory to Cobbler, copy this script to `/etc/ansible` and `chmod +x` the file. Run `cobblerd` any time you use Ansible and use the `-i` command line option (for example, `-i /etc/ansible/cobbler.py`) to communicate with Cobbler using Cobbler's XMLRPC API.

Add a `cobbler.ini` file in `/etc/ansible` so Ansible knows where the Cobbler server is and some cache improvements can be used. For example:

```
[cobbler]

# Set Cobbler's hostname or IP address
host = http://127.0.0.1/cobbler_api

# API calls to Cobbler can be slow. For this reason, we cache the results of an API
# call. Set this to the path you want cache files to be written to. Two files
# will be written to this directory:
#   - ansible-cobbler.cache
#   - ansible-cobbler.index

cache_path = /tmp

# The number of seconds a cache file is considered valid. After this many
# seconds, a new API call will be made, and the cache file will be updated.

cache_max_age = 900
```

First test the script by running `/etc/ansible/cobbler.py` directly. You should see some JSON data output, but it may not have anything in it just yet.

Let's explore what this does. In Cobbler, assume a scenario somewhat like the following:

```
cobbler profile add --name=webserver --distro=CentOS6-x86_64
cobbler profile edit --name=webserver --mgmt-classes="webserver" --ksmeta="a=2 b=3"
cobbler system edit --name=foo --dns-name="foo.example.com" --mgmt-classes="atlanta" --
↪ksmeta="c=4"
cobbler system edit --name=bar --dns-name="bar.example.com" --mgmt-classes="atlanta" --
↪ksmeta="c=5"
```

In the example above, the system 'foo.example.com' is addressable by ansible directly, but is also addressable when using the group names 'webserver' or 'atlanta'. Since Ansible uses SSH, it contacts system foo over 'foo.example.com', only, never just 'foo'. Similarly, if you tried "ansible foo", it would not find the system… but "ansible 'foo*'" would do, because the system DNS name starts with 'foo'.

The script provides more than host and group info. In addition, as a bonus, when the 'setup' module is run (which happens automatically when using playbooks), the variables 'a', 'b', and 'c' will all be auto-populated in the templates:

```
# file: /srv/motd.j2
Welcome, I am templated with a value of a={{ a }}, b={{ b }}, and c={{ c }}
```

Which could be executed just like this:

```
ansible webserver -m setup
ansible webserver -m template -a "src=/tmp/motd.j2 dest=/etc/motd"
```

Note: The name 'webserver' came from Cobbler, as did the variables for the config file. You can still pass in your own variables like normal in Ansible, but variables from the external inventory script will override any that have the same name.

So, with the template above (`motd.j2`), this results in the following data being written to `/etc/motd` for system 'foo':

```
Welcome, I am templated with a value of a=2, b=3, and c=4
```

And on system 'bar' (bar.example.com):

```
Welcome, I am templated with a value of a=2, b=3, and c=5
```

And technically, though there is no major good reason to do it, this also works:

```
ansible webserver -m ansible.builtin.shell -a "echo {{ a }}"
```

So, in other words, you can use those variables in arguments/actions as well.

Inventory script example: OpenStack

If you use an OpenStack-based cloud, instead of manually maintaining your own inventory file, you can use the `openstack_inventory.py` dynamic inventory to pull information about your compute instances directly from OpenStack.

You can download the latest version of the OpenStack inventory script here.

You can use the inventory script explicitly (by passing the *-i openstack_inventory.py* argument to Ansible) or implicitly (by placing the script at */etc/ansible/hosts*).

Explicit use of OpenStack inventory script

Download the latest version of the OpenStack dynamic inventory script and make it executable.

```
wget https://raw.githubusercontent.com/openstack/ansible-collections-openstack/master/
↪scripts/inventory/openstack_inventory.py
chmod +x openstack_inventory.py
```

Note: Do not name it *openstack.py*. This name will conflict with imports from openstacksdk.

Source an OpenStack RC file:

```
source openstack.rc
```

Note: An OpenStack RC file contains the environment variables required by the client tools to establish a connection with the cloud provider, such as the authentication URL, user name, password and region name. For more information on how to download, create or source an OpenStack RC file, please refer to Set environment variables using the OpenStack RC file.

You can confirm the file has been successfully sourced by running a simple command, such as *nova list* and ensuring it returns no errors.

Note: The OpenStack command line clients are required to run the *nova list* command. For more information on how to install them, please refer to Install the OpenStack command-line clients.

You can test the OpenStack dynamic inventory script manually to confirm it is working as expected:

```
./openstack_inventory.py --list
```

After a few moments you should see some JSON output with information about your compute instances.

Once you confirm the dynamic inventory script is working as expected, you can tell Ansible to use the *openstack_inventory.py* script as an inventory file, as illustrated below:

```
ansible -i openstack_inventory.py all -m ansible.builtin.ping
```

Implicit use of OpenStack inventory script

Download the latest version of the OpenStack dynamic inventory script, make it executable and copy it to */etc/ansible/hosts*:

```
wget https://raw.githubusercontent.com/openstack/ansible-collections-openstack/master/
↪scripts/inventory/openstack_inventory.py
chmod +x openstack_inventory.py
sudo cp openstack_inventory.py /etc/ansible/hosts
```

Download the sample configuration file, modify it to suit your needs and copy it to */etc/ansible/openstack.yml*:

```
wget https://raw.githubusercontent.com/openstack/ansible-collections-openstack/master/
↪scripts/inventory/openstack.yml
vi openstack.yml
sudo cp openstack.yml /etc/ansible/
```

You can test the OpenStack dynamic inventory script manually to confirm it is working as expected:

```
/etc/ansible/hosts --list
```

After a few moments you should see some JSON output with information about your compute instances.

Refreshing the cache

Note that the OpenStack dynamic inventory script will cache results to avoid repeated API calls. To explicitly clear the cache, you can run the openstack_inventory.py (or hosts) script with the `--refresh` parameter:

```
./openstack_inventory.py --refresh --list
```

Other inventory scripts

In Ansible 2.10 and later, inventory scripts moved to their associated collections. Many are now in the ansible-community/contrib-scripts repository. We recommend you use *Inventory plugins* instead.

Using inventory directories and multiple inventory sources

If the location given to `-i` in Ansible is a directory (or as so configured in `ansible.cfg`), Ansible can use multiple inventory sources at the same time. When doing so, it is possible to mix both dynamic and statically managed inventory sources in the same ansible run. Instant hybrid cloud!

In an inventory directory, executable files are treated as dynamic inventory sources and most other files as static sources. Files which end with any of the following are ignored:

```
~, .orig, .bak, .ini, .cfg, .retry, .pyc, .pyo
```

You can replace this list with your own selection by configuring an `inventory_ignore_extensions` list in `ansible.cfg`, or setting the *`ANSIBLE_INVENTORY_IGNORE`* environment variable. The value in either case must be a comma-separated list of patterns, as shown above.

Any `group_vars` and `host_vars` subdirectories in an inventory directory are interpreted as expected, making inventory directories a powerful way to organize different sets of configurations. See *Passing multiple inventory sources* for more information.

Static groups of dynamic groups

When defining groups of groups in the static inventory file, the child groups must also be defined in the static inventory file, otherwise ansible returns an error. If you want to define a static group of dynamic child groups, define the dynamic groups as empty in the static inventory file. For example:

```
[tag_Name_staging_foo]

[tag_Name_staging_bar]

[staging:children]
tag_Name_staging_foo
tag_Name_staging_bar
```

See also:

How to build your inventory
All about static inventory files

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.4.3 Patterns: targeting hosts and groups

When you execute Ansible through an ad hoc command or by running a playbook, you must choose which managed nodes or groups you want to execute against. Patterns let you run commands and playbooks against specific hosts and/or groups in your inventory. An Ansible pattern can refer to a single host, an IP address, an inventory group, a set of groups, or all hosts in your inventory. Patterns are highly flexible - you can exclude or require subsets of hosts, use wildcards or regular expressions, and more. Ansible executes on all inventory hosts included in the pattern.

Using patterns

You use a pattern almost any time you execute an ad hoc command or a playbook. The pattern is the only element of an *ad hoc command* that has no flag. It is usually the second element:

```
ansible <pattern> -m <module_name> -a "<module options>"
```

For example:

```
ansible webservers -m service -a "name=httpd state=restarted"
```

In a playbook the pattern is the content of the `hosts:` line for each play:

```
- name: <play_name>
  hosts: <pattern>
```

For example:

```
- name: restart webservers
  hosts: webservers
```

Since you often want to run a command or playbook against multiple hosts at once, patterns often refer to inventory groups. Both the ad hoc command and the playbook above will execute against all machines in the `webservers` group.

Common patterns

This table lists common patterns for targeting inventory hosts and groups.

Description	Pattern(s)	Targets
All hosts	all (or *)	
One host	host1	
Multiple hosts	host1:host2 (or host1,host2)	
One group	webservers	
Multiple groups	webservers:dbservers	all hosts in webservers plus all hosts in dbservers
Excluding groups	webservers:!atlanta	all hosts in webservers except those in atlanta
Intersection of groups	webservers:&staging	any hosts in webservers that are also in staging

Note: You can use either a comma (`,`) or a colon (`:`) to separate a list of hosts. The comma is preferred when dealing with ranges and IPv6 addresses.

Once you know the basic patterns, you can combine them. This example:

```
webservers:dbservers:&staging:!phoenix
```

targets all machines in the groups 'webservers' and 'dbservers' that are also in the group 'staging', except any machines in the group 'phoenix'.

You can use wildcard patterns with FQDNs or IP addresses, as long as the hosts are named in your inventory by FQDN or IP address:

```
192.0.*
*.example.com
*.com
```

You can mix wildcard patterns and groups at the same time:

```
one*.com:dbservers
```

Limitations of patterns

Patterns depend on inventory. If a host or group is not listed in your inventory, you cannot use a pattern to target it. If your pattern includes an IP address or hostname that does not appear in your inventory, you will see an error like this:

```
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: Could not match supplied host pattern, ignoring: *.not_in_inventory.com
```

Your pattern must match your inventory syntax. If you define a host as an *alias*:

```
atlanta:
  host1:
    http_port: 80
    maxRequestsPerChild: 808
    host: 127.0.0.2
```

you must use the alias in your pattern. In the example above, you must use `host1` in your pattern. If you use the IP address, you will once again get the error:

```
[WARNING]: Could not match supplied host pattern, ignoring: 127.0.0.2
```

Pattern processing order

The processing is a bit special and happens in the following order:

1. `:` and `,`
2. `&`
3. `!`

This positioning only accounts for processing order inside each operation: `a:b:&c:!d:!e == &c:a:!d:b:!e == !d:a:!e:&c:b`

All of these result in the following:

Host in/is (a or b) AND host in/is all(c) AND host NOT in/is all(d, e).

Now `a:b:!e:!d:&c` is a slight change as the `!e` gets processed before the `!d`, though this doesn't make much of a difference:

Host in/is (a or b) AND host in/is all(c) AND host NOT in/is all(e, d).

Advanced pattern options

The common patterns described above will meet most of your needs, but Ansible offers several other ways to define the hosts and groups you want to target.

Using variables in patterns

You can use variables to enable passing group specifiers via the `-e` argument to ansible-playbook:

```
webservers:!{{ excluded }}:&{{ required }}
```

Using group position in patterns

You can define a host or subset of hosts by its position in a group. For example, given the following group:

```
[webservers]
cobweb
webbing
weber
```

you can use subscripts to select individual hosts or ranges within the webservers group:

```
webservers[0]       # == cobweb
webservers[-1]      # == weber
webservers[0:2]     # == webservers[0],webservers[1]
                    # == cobweb,webbing
webservers[1:]      # == webbing,weber
webservers[:3]      # == cobweb,webbing,weber
```

Using regexes in patterns

You can specify a pattern as a regular expression by starting the pattern with ~:

```
~(web|db).*\.example\.com
```

Patterns and ad-hoc commands

You can change the behavior of the patterns defined in ad-hoc commands using command-line options. You can also limit the hosts you target on a particular run with the `--limit` flag.

- Limit to one host

```
$ ansible all -m [module] -a "[module options]" --limit "host1"
```

- Limit to multiple hosts

```
$ ansible all -m [module] -a "[module options]" --limit "host1,host2"
```

- Negated limit. Note that single quotes MUST be used to prevent bash interpolation.

```
$ ansible all -m [module] -a "[module options]" --limit 'all:!host1'
```

- Limit to host group

```
$ ansible all -m [module] -a "[module options]" --limit 'group1'
```

Patterns and ansible-playbook flags

You can change the behavior of the patterns defined in playbooks using command-line options. For example, you can run a playbook that defines `hosts: all` on a single host by specifying `-i 127.0.0.2,` (note the trailing comma). This works even if the host you target is not defined in your inventory, but this method will NOT read your inventory for variables tied to this host and any variables required by the playbook will need to be specified manually at the command line. You can also limit the hosts you target on a particular run with the `--limit` flag, which will reference your inventory:

```
ansible-playbook site.yml --limit datacenter2
```

Finally, you can use `--limit` to read the list of hosts from a file by prefixing the file name with `@`:

```
ansible-playbook site.yml --limit @retry_hosts.txt
```

If *RETRY_FILES_ENABLED* is set to `True`, a `.retry` file will be created after the `ansible-playbook` run containing a list of failed hosts from all plays. This file is overwritten each time `ansible-playbook` finishes running.

```
ansible-playbook site.yml --limit @site.retry
```

To apply your knowledge of patterns with Ansible commands and playbooks, read *Introduction to ad hoc commands* and *Ansible playbooks*.

See also:

Introduction to ad hoc commands
Examples of basic commands

Working with playbooks
Learning the Ansible configuration management language

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.4.4 Connection methods and details

This section shows you how to expand and refine the connection methods Ansible uses for your inventory.

ControlPersist and paramiko

By default, Ansible uses native OpenSSH, because it supports ControlPersist (a performance feature), Kerberos, and options in `~/.ssh/config` such as Jump Host setup. If your control machine uses an older version of OpenSSH that does not support ControlPersist, Ansible will fallback to a Python implementation of OpenSSH called 'paramiko'.

Setting a remote user

By default, Ansible connects to all remote devices with the user name you are using on the control node. If that user name does not exist on a remote device, you can set a different user name for the connection. If you just need to do some tasks as a different user, look at *Understanding privilege escalation: become*. You can set the connection user in a playbook:

```
---
- name: update webservers
  hosts: webservers
  remote_user: admin

  tasks:
  - name: thing to do first in this playbook
  . . .
```

as a host variable in inventory:

```
other1.example.com     ansible_connection=ssh        ansible_user=myuser
other2.example.com     ansible_connection=ssh        ansible_user=myotheruser
```

or as a group variable in inventory:

```
cloud:
  hosts:
    cloud1: my_backup.cloud.com
    cloud2: my_backup2.cloud.com
  vars:
    ansible_user: admin
```

See also:

ssh_connection
    Details on the `remote_user` keyword and `ansible_user` variable.

Controlling how Ansible behaves: precedence rules
    Details on Ansible precedence rules.

Setting up SSH keys

By default, Ansible assumes you are using SSH keys to connect to remote machines. SSH keys are encouraged, but you can use password authentication if needed with the `--ask-pass` option. If you need to provide a password for *privilege escalation* (sudo, pbrun, and so on), use `--ask-become-pass`.

Note: Ansible does not expose a channel to allow communication between the user and the ssh process to accept a password manually to decrypt an ssh key when using the ssh connection plugin (which is the default). The use of `ssh-agent` is highly recommended.

To set up SSH agent to avoid retyping passwords, you can do:

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

Depending on your setup, you may wish to use Ansible's `--private-key` command line option to specify a pem file instead. You can also add the private key file:

```
$ ssh-agent bash
$ ssh-add ~/.ssh/keypair.pem
```

Another way to add private key files without using ssh-agent is using `ansible_ssh_private_key_file` in an inventory file as explained here: *How to build your inventory*.

Running against localhost

You can run commands against the control node by using "localhost" or "127.0.0.1" for the server name:

```
$ ansible localhost -m ping -e 'ansible_python_interpreter="/usr/bin/env python"'
```

You can specify localhost explicitly by adding this to your inventory file:

```
localhost ansible_connection=local ansible_python_interpreter="/usr/bin/env python"
```

Managing host key checking

Ansible enables host key checking by default. Checking host keys guards against server spoofing and man-in-the-middle attacks, but it does require some maintenance.

If a host is reinstalled and has a different key in 'known_hosts', this will result in an error message until corrected. If a new host is not in 'known_hosts' your control node may prompt for confirmation of the key, which results in an interactive experience if using Ansible, from say, cron. You might not want this.

If you understand the implications and wish to disable this behavior, you can do so by editing `/etc/ansible/ansible.cfg` or `~/.ansible.cfg`:

```
[defaults]
host_key_checking = False
```

Alternatively this can be set by the *`ANSIBLE_HOST_KEY_CHECKING`* environment variable:

```
$ export ANSIBLE_HOST_KEY_CHECKING=False
```

Also note that host key checking in paramiko mode is reasonably slow, therefore switching to 'ssh' is also recommended when using this feature.

Other connection methods

Ansible can use a variety of connection methods beyond SSH. You can select any connection plugin, including managing things locally and managing chroot, lxc, and jail containers. A mode called 'ansible-pull' can also invert the system and have systems 'phone home' via scheduled git checkouts to pull configuration directives from a central repository.

1.5 Using Ansible command line tools

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the guide for using Ansible command line tools. Ansible provides ad hoc commands and several utilities for performing various operations and automation tasks.

1.5.1 Introduction to ad hoc commands

An Ansible ad hoc command uses the */usr/bin/ansible* command-line tool to automate a single task on one or more managed nodes. ad hoc commands are quick and easy, but they are not reusable. So why learn about ad hoc commands? ad hoc commands demonstrate the simplicity and power of Ansible. The concepts you learn here will port over directly to the playbook language. Before reading and executing these examples, please read *How to build your inventory*.

Why use ad hoc commands?

ad hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad hoc command looks like this:

```
$ ansible [pattern] -m [module] -a "[module options]"
```

The `-a` option accepts options either through the `key=value` syntax or a JSON string starting with `{` and ending with `}` for more complex option structure. You can learn more about *patterns* and modules on other pages.

Use cases for ad hoc tasks

ad hoc tasks can be used to reboot servers, copy files, manage packages and users, and much more. You can use any Ansible module in an ad hoc task. ad hoc tasks, like playbooks, use a declarative model, calculating and executing the actions required to reach a specified final state. They achieve a form of idempotence by checking the current state before they begin and doing nothing unless the current state is different from the specified final state.

Rebooting servers

The default module for the `ansible` command-line utility is the ansible.builtin.command module. You can use an ad hoc task to call the command module and reboot all web servers in Atlanta, 10 at a time. Before Ansible can do this, you must have all servers in Atlanta listed in a group called [atlanta] in your inventory, and you must have working SSH credentials for each machine in that group. To reboot all the servers in the [atlanta] group:

```
$ ansible atlanta -a "/sbin/reboot"
```

By default Ansible uses only 5 simultaneous processes. If you have more hosts than the value set for the fork count, Ansible will talk to them, but it will take a little longer. To reboot the [atlanta] servers with 10 parallel forks:

```
$ ansible atlanta -a "/sbin/reboot" -f 10
```

/usr/bin/ansible will default to running from your user account. To connect as a different user:

```
$ ansible atlanta -a "/sbin/reboot" -f 10 -u username
```

Rebooting probably requires privilege escalation. You can connect to the server as `username` and run the command as the `root` user by using the *become* keyword:

```
$ ansible atlanta -a "/sbin/reboot" -f 10 -u username --become [--ask-become-pass]
```

If you add `--ask-become-pass` or `-K`, Ansible prompts you for the password to use for privilege escalation (sudo/su/pfexec/doas/etc).

Note: The command module does not support extended shell syntax like piping and redirects (although shell variables will always work). If your command requires shell-specific syntax, use the *shell* module instead. Read more about the differences on the Working With Modules page.

So far all our examples have used the default 'command' module. To use a different module, pass `-m` for module name. For example, to use the ansible.builtin.shell module:

```
$ ansible raleigh -m ansible.builtin.shell -a 'echo $TERM'
```

When running any command with the Ansible *ad hoc* CLI (as opposed to *Playbooks*), pay particular attention to shell quoting rules, so the local shell retains the variable and passes it to Ansible. For example, using double rather than single quotes in the above example would evaluate the variable on the box you were on.

Managing files

An ad hoc task can harness the power of Ansible and SCP to transfer many files to multiple machines in parallel. To transfer a file directly to all servers in the [atlanta] group:

```
$ ansible atlanta -m ansible.builtin.copy -a "src=/etc/hosts dest=/tmp/hosts"
```

If you plan to repeat a task like this, use the ansible.builtin.template module in a playbook.

The ansible.builtin.file module allows changing ownership and permissions on files. These same options can be passed directly to the `copy` module as well:

```
$ ansible webservers -m ansible.builtin.file -a "dest=/srv/foo/a.txt mode=600"
$ ansible webservers -m ansible.builtin.file -a "dest=/srv/foo/b.txt mode=600␣
↪owner=mdehaan group=mdehaan"
```

The `file` module can also create directories, similar to `mkdir -p`:

```
$ ansible webservers -m ansible.builtin.file -a "dest=/path/to/c mode=755 owner=mdehaan␣
↪group=mdehaan state=directory"
```

As well as delete directories (recursively) and delete files:

```
$ ansible webservers -m ansible.builtin.file -a "dest=/path/to/c state=absent"
```

Managing packages

You might also use an ad hoc task to install, update, or remove packages on managed nodes using a package management module such as `yum`. Package management modules support common functions to install, remove, and generally manage packages. Some specific functions for a package manager might not be present in the Ansible module since they are not part of general package management.

To ensure a package is installed without updating it:

```
$ ansible webservers -m ansible.builtin.yum -a "name=acme state=present"
```

To ensure a specific version of a package is installed:

```
$ ansible webservers -m ansible.builtin.yum -a "name=acme-1.5 state=present"
```

To ensure a package is at the latest version:

```
$ ansible webservers -m ansible.builtin.yum -a "name=acme state=latest"
```

To ensure a package is not installed:

```
$ ansible webservers -m ansible.builtin.yum -a "name=acme state=absent"
```

Ansible has modules for managing packages under many platforms. If there is no module for your package manager, you can install packages using the command module or create a module for your package manager.

Managing users and groups

You can create, manage, and remove user accounts on your managed nodes with ad hoc tasks:

```
$ ansible all -m ansible.builtin.user -a "name=foo password=<crypted password here>"

$ ansible all -m ansible.builtin.user -a "name=foo state=absent"
```

See the ansible.builtin.user module documentation for details on all of the available options, including how to manipulate groups and group membership.

Managing services

Ensure a service is started on all webservers:

```
$ ansible webservers -m ansible.builtin.service -a "name=httpd state=started"
```

Alternatively, restart a service on all webservers:

```
$ ansible webservers -m ansible.builtin.service -a "name=httpd state=restarted"
```

Ensure a service is stopped:

```
$ ansible webservers -m ansible.builtin.service -a "name=httpd state=stopped"
```

Gathering facts

Facts represent discovered variables about a system. You can use facts to implement conditional execution of tasks but also just to get ad hoc information about your systems. To see all facts:

```
$ ansible all -m ansible.builtin.setup
```

You can also filter this output to display only certain facts, see the ansible.builtin.setup module documentation for details.

Patterns and ad-hoc commands

See the *patterns* documentation for details on all of the available options, including how to limit using patterns in ad-hoc commands.

Now that you understand the basic elements of Ansible execution, you are ready to learn to automate repetitive tasks using *Ansible Playbooks*.

See also:

Configuring Ansible
All about the Ansible config file

Collection Index
Browse existing collections, modules, and plugins

Working with playbooks
Using Ansible for configuration management & deployment

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.5.2 Working with command line tools

Most users are familiar with *ansible* and *ansible-playbook*, but those are not the only utilities Ansible provides. Below is a complete list of Ansible utilities. Each page contains a description of the utility and a listing of supported parameters.

Note: You should not run most Ansible CLI tools in parallel against the same targets.

ansible

Define and run a single task 'playbook' against a set of hosts

Synopsis

```
usage: ansible [-h] [--version] [-v] [-b] [--become-method BECOME_METHOD] [--become-user␣
↪BECOME_USER] [-K | --become-password-file BECOME_PASSWORD_FILE] [-i INVENTORY] [--list-
↪hosts] [-l SUBSET]
            [-P POLL_INTERVAL] [-B SECONDS] [-o] [-t TREE] [--private-key PRIVATE_KEY_
↪FILE] [-u REMOTE_USER] [-c CONNECTION] [-T TIMEOUT] [--ssh-common-args SSH_COMMON_ARGS]
            [--sftp-extra-args SFTP_EXTRA_ARGS] [--scp-extra-args SCP_EXTRA_ARGS] [--ssh-
↪extra-args SSH_EXTRA_ARGS] [-k | --connection-password-file CONNECTION_PASSWORD_FILE]␣
↪[-C] [-D] [-e EXTRA_VARS]
            [--vault-id VAULT_IDS] [--ask-vault-password | --vault-password-file VAULT_
↪PASSWORD_FILES] [-f FORKS] [-M MODULE_PATH] [--playbook-dir BASEDIR] [--task-timeout␣
↪TASK_TIMEOUT] [-a MODULE_ARGS]
            [-m MODULE_NAME]
            pattern
```

Description

is an extra-simple tool/framework/API for doing 'remote things'. this command allows you to define and run a single task 'playbook' against a set of hosts

Common Options

`--ask-vault-password`, **`--ask-vault-pass`**
: ask for vault password

`--become-method` `<BECOME_METHOD>`
: privilege escalation method to use (default=sudo), use *ansible-doc -t become -l* to list valid choices.

`--become-password-file` `<BECOME_PASSWORD_FILE>`, **`--become-pass-file`** `<BECOME_PASSWORD_FILE>`
: Become password file

`--become-user` `<BECOME_USER>`
: run operations as this user (default=root)

`--connection-password-file` `<CONNECTION_PASSWORD_FILE>`, **`--conn-pass-file`** `<CONNECTION_PASSWORD_FILE>`
: Connection password file

`--list-hosts`
: outputs a list of matching hosts; does not execute anything else

`--playbook-dir` `<BASEDIR>`
: Since this tool does not use playbooks, use this as a substitute playbook directory. This sets the relative path for many features including roles/ group_vars/ etc.

`--private-key` `<PRIVATE_KEY_FILE>`, **`--key-file`** `<PRIVATE_KEY_FILE>`
: use this file to authenticate the connection

`--scp-extra-args` `<SCP_EXTRA_ARGS>`
: specify extra arguments to pass to scp only (e.g. -l)

`--sftp-extra-args` `<SFTP_EXTRA_ARGS>`
: specify extra arguments to pass to sftp only (e.g. -f, -l)

`--ssh-common-args` `<SSH_COMMON_ARGS>`
: specify common arguments to pass to sftp/scp/ssh (e.g. ProxyCommand)

`--ssh-extra-args` `<SSH_EXTRA_ARGS>`
: specify extra arguments to pass to ssh only (e.g. -R)

`--task-timeout` `<TASK_TIMEOUT>`
: set task timeout limit in seconds, must be positive integer.

`--vault-id`
: the vault identity to use

`--vault-password-file`, **`--vault-pass-file`**
: vault password file

`--version`
: show program's version number, config file location, configured module search path, module location, executable location and exit

`-B <SECONDS>, --background <SECONDS>`
: run asynchronously, failing after X seconds (default=N/A)

`-C, --check`
: don't make any changes; instead, try to predict some of the changes that may occur

`-D, --diff`
: when changing (small) files and templates, show the differences in those files; works great with –check

`-K, --ask-become-pass`
: ask for privilege escalation password

`-M, --module-path`
: prepend colon-separated path(s) to module library (default={{ ANSIBLE_HOME ~ "/plugins/modules:/usr/share/ansible/plugins/modules" }})

`-P <POLL_INTERVAL>, --poll <POLL_INTERVAL>`
: set the poll interval if using -B (default=15)

`-T <TIMEOUT>, --timeout <TIMEOUT>`
: override the connection timeout in seconds (default=10)

`-a <MODULE_ARGS>, --args <MODULE_ARGS>`
: The action's options in space separated k=v format: -a 'opt1=val1 opt2=val2' or a json string: -a '{"opt1": "val1", "opt2": "val2"}'

`-b, --become`
: run operations with become (does not imply password prompting)

`-c <CONNECTION>, --connection <CONNECTION>`
: connection type to use (default=smart)

`-e, --extra-vars`
: set additional variables as key=value or YAML/JSON, if filename prepend with @

`-f <FORKS>, --forks <FORKS>`
: specify number of parallel processes to use (default=5)

`-h, --help`
: show this help message and exit

`-i, --inventory, --inventory-file`
: specify inventory host path or comma separated host list. –inventory-file is deprecated

`-k, --ask-pass`
: ask for connection password

`-l <SUBSET>, --limit <SUBSET>`
: further limit selected hosts to an additional pattern

`-m <MODULE_NAME>, --module-name <MODULE_NAME>`
: Name of the action to execute (default=command)

-o, --one-line
condense output

-t <TREE>, **--tree** <TREE>
log output to this directory

-u <REMOTE_USER>, **--user** <REMOTE_USER>
connect as this user (default=None)

-v, --verbose
Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-config

View ansible configuration.

Synopsis

```
usage: ansible-config [-h] [--version] [-v] {list,dump,view,init} ...
```

Description

Config command line class

Common Options

`--version`
show program's version number, config file location, configured module search path, module location, executable location and exit

`-h, --help`
show this help message and exit

`-v, --verbose`
Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Actions

list

list and output available configs

--format <FORMAT>, **-f** <FORMAT>
: Output format for list

-c <CONFIG_FILE>, **--config** <CONFIG_FILE>
: path to configuration file, defaults to first file found in precedence.

-t <TYPE>, **--type** <TYPE>
: Filter down to a specific plugin type.

dump

Shows the current settings, merges ansible.cfg if specified

--format <FORMAT>, **-f** <FORMAT>
: Output format for dump

--only-changed, **--changed-only**
: Only show configurations that have changed from the default

-c <CONFIG_FILE>, **--config** <CONFIG_FILE>
: path to configuration file, defaults to first file found in precedence.

-t <TYPE>, **--type** <TYPE>
: Filter down to a specific plugin type.

view

Displays the current config file

-c <CONFIG_FILE>, **--config** <CONFIG_FILE>
: path to configuration file, defaults to first file found in precedence.

-t <TYPE>, **--type** <TYPE>
: Filter down to a specific plugin type.

init

--disabled
: Prefixes all entries with a comment character to disable them

--format <FORMAT>, **-f** <FORMAT>
: Output format for init

-c <CONFIG_FILE>, **--config** <CONFIG_FILE>
: path to configuration file, defaults to first file found in precedence.

-t <TYPE>, **--type** <TYPE>
: Filter down to a specific plugin type.

Environment

The following environment variables may be specified.

`ANSIBLE_CONFIG` – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-console

REPL console for executing Ansible tasks.

Synopsis

```
usage: ansible-console [-h] [--version] [-v] [-b] [--become-method BECOME_METHOD] [--
↪become-user BECOME_USER] [-K | --become-password-file BECOME_PASSWORD_FILE] [-i␣
↪INVENTORY] [--list-hosts] [-l SUBSET]
                    [--private-key PRIVATE_KEY_FILE] [-u REMOTE_USER] [-c CONNECTION] [-
↪T TIMEOUT] [--ssh-common-args SSH_COMMON_ARGS] [--sftp-extra-args SFTP_EXTRA_ARGS] [--
↪scp-extra-args SCP_EXTRA_ARGS]
                    [--ssh-extra-args SSH_EXTRA_ARGS] [-k | --connection-password-file␣
↪CONNECTION_PASSWORD_FILE] [-C] [-D] [--vault-id VAULT_IDS]
                    [--ask-vault-password | --vault-password-file VAULT_PASSWORD_FILES]␣
↪[-f FORKS] [-M MODULE_PATH] [--playbook-dir BASEDIR] [-e EXTRA_VARS] [--task-timeout␣
↪TASK_TIMEOUT] [--step]
                    [pattern]
```

Description

A REPL that allows for running ad-hoc tasks against a chosen inventory from a nice shell with built-in tab completion (based on dominis' ansible-shell).

It supports several commands, and you can modify its configuration at runtime:

- *cd [pattern]*: change host/group (you can use host patterns eg.: app*.dc*:!app01*)
- *list*: list available hosts in the current path
- *list groups*: list groups included in the current path
- *become*: toggle the become flag
- *!*: forces shell module instead of the ansible module (!yum update -y)
- *verbosity [num]*: set the verbosity level
- *forks [num]*: set the number of forks
- *become_user [user]*: set the become_user
- *remote_user [user]*: set the remote_user
- *become_method [method]*: set the privilege escalation method
- *check [bool]*: toggle check mode
- *diff [bool]*: toggle diff mode
- *timeout [integer]*: set the timeout of tasks in seconds (0 to disable)
- *help [command/module]*: display documentation for the command or module
- *exit*: exit ansible-console

Common Options

--ask-vault-password, --ask-vault-pass
: ask for vault password

--become-method <BECOME_METHOD>
: privilege escalation method to use (default=sudo), use *ansible-doc -t become -l* to list valid choices.

--become-password-file <BECOME_PASSWORD_FILE>, **--become-pass-file** <BECOME_PASSWORD_FILE>
: Become password file

--become-user <BECOME_USER>
: run operations as this user (default=root)

--connection-password-file <CONNECTION_PASSWORD_FILE>, **--conn-pass-file** <CONNECTION_PASSWORD_FILE>
: Connection password file

--list-hosts
: outputs a list of matching hosts; does not execute anything else

--playbook-dir <BASEDIR>
: Since this tool does not use playbooks, use this as a substitute playbook directory. This sets the relative path for many features including roles/ group_vars/ etc.

--private-key <PRIVATE_KEY_FILE>, **--key-file** <PRIVATE_KEY_FILE>
: use this file to authenticate the connection

--scp-extra-args <SCP_EXTRA_ARGS>
: specify extra arguments to pass to scp only (e.g. -l)

--sftp-extra-args <SFTP_EXTRA_ARGS>
: specify extra arguments to pass to sftp only (e.g. -f, -l)

--ssh-common-args <SSH_COMMON_ARGS>
: specify common arguments to pass to sftp/scp/ssh (e.g. ProxyCommand)

--ssh-extra-args <SSH_EXTRA_ARGS>
: specify extra arguments to pass to ssh only (e.g. -R)

--step
: one-step-at-a-time: confirm each task before running

--task-timeout <TASK_TIMEOUT>
: set task timeout limit in seconds, must be positive integer.

--vault-id
: the vault identity to use

--vault-password-file, --vault-pass-file
: vault password file

--version
: show program's version number, config file location, configured module search path, module location, executable location and exit

-C, --check
: don't make any changes; instead, try to predict some of the changes that may occur

-D, --diff
: when changing (small) files and templates, show the differences in those files; works great with –check

-K, --ask-become-pass
: ask for privilege escalation password

-M, --module-path
: prepend colon-separated path(s) to module library (default={{ ANSIBLE_HOME ~ "/plugins/modules:/usr/share/ansible/plugins/modules" }})

-T <TIMEOUT>, **--timeout** <TIMEOUT>
: override the connection timeout in seconds (default=10)

-b, --become
: run operations with become (does not imply password prompting)

-c <CONNECTION>, **--connection** <CONNECTION>
: connection type to use (default=smart)

-e, --extra-vars
: set additional variables as key=value or YAML/JSON, if filename prepend with @

-f <FORKS>, **--forks** <FORKS>
: specify number of parallel processes to use (default=5)

-h, --help
: show this help message and exit

-i, --inventory, --inventory-file
: specify inventory host path or comma separated host list. –inventory-file is deprecated

-k, --ask-pass
: ask for connection password

-l <SUBSET>, **--limit** <SUBSET>
: further limit selected hosts to an additional pattern

-u <REMOTE_USER>, **--user** <REMOTE_USER>
: connect as this user (default=None)

-v, --verbose
: Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-doc

plugin documentation tool

Synopsis

```
usage: ansible-doc [-h] [--version] [-v] [-M MODULE_PATH] [--playbook-dir BASEDIR]
                [-t {become,cache,callback,cliconf,connection,httpapi,inventory,lookup,
↪netconf,shell,vars,module,strategy,test,filter,role,keyword}] [-j] [-r ROLES_PATH]
                [-e ENTRY_POINT | -s | -F | -l | --metadata-dump] [--no-fail-on-errors]
                [plugin ...]
```

Description

displays information on modules installed in Ansible libraries. It displays a terse listing of plugins and their short descriptions, provides a printout of their DOCUMENTATION strings, and it can create a short "snippet" which can be pasted into a playbook.

Common Options

`--metadata-dump`
: **For internal use only** Dump json metadata for all entries, ignores other options.

`--no-fail-on-errors`
: **For internal use only** Only used for –metadata-dump. Do not fail on errors. Report the error message in the JSON instead.

`--playbook-dir` `<BASEDIR>`
: Since this tool does not use playbooks, use this as a substitute playbook directory. This sets the relative path for many features including roles/ group_vars/ etc.

`--version`
: show program's version number, config file location, configured module search path, module location, executable location and exit

`-F`, **`--list_files`**
: Show plugin names and their source files without summaries (implies –list). A supplied argument will be used for filtering, can be a namespace or full collection name.

`-M`, **`--module-path`**
: prepend colon-separated path(s) to module library (default={{ ANSIBLE_HOME ~ "/plugins/modules:/usr/share/ansible/plugins/modules" }})

`-e` `<ENTRY_POINT>`, **`--entry-point`** `<ENTRY_POINT>`
: Select the entry point for role(s).

`-h`, **`--help`**
: show this help message and exit

`-j`, **`--json`**
: Change output into json format.

`-l`, **`--list`**
: List available plugins. A supplied argument will be used for filtering, can be a namespace or full collection name.

`-r`, **`--roles-path`**
: The path to the directory containing your roles.

`-s`, **`--snippet`**
: Show playbook snippet for these plugin types: inventory, lookup, module

`-t` `<TYPE>`, **`--type`** `<TYPE>`
: Choose which plugin type (defaults to "module"). Available plugin types are : ('become', 'cache', 'callback', 'cliconf', 'connection', 'httpapi', 'inventory', 'lookup', 'netconf', 'shell', 'vars', 'module', 'strategy', 'test', 'filter', 'role', 'keyword')

`-v`, **`--verbose`**
: Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-galaxy

Perform various Role and Collection related operations.

Synopsis

```
usage: ansible-galaxy [-h] [--version] [-v] TYPE ...
```

Description

Command to manage Ansible roles and collections.

None of the CLI tools are designed to run concurrently with themselves. Use an external scheduler and/or locking to ensure there are no clashing operations.

Common Options

`--version`
show program's version number, config file location, configured module search path, module location, executable location and exit

`-h, --help`
show this help message and exit

`-v, --verbose`
Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Actions

collection

Perform the action on an Ansible Galaxy collection. Must be combined with a further action like init/install as listed below.

collection download

--clear-response-cache
: Clear the existing server response cache.

--no-cache
: Do not use the server response cache.

--pre
: Include pre-release versions. Semantic versioning pre-releases are ignored by default

--timeout `<TIMEOUT>`
: The time to wait for operations against the galaxy server, defaults to 60s.

--token `<API_KEY>`, **--api-key** `<API_KEY>`
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-n, --no-deps
: Don't download collection(s) listed as dependencies.

-p `<DOWNLOAD_PATH>`, **--download-path** `<DOWNLOAD_PATH>`
: The directory to download the collections to.

-r `<REQUIREMENTS>`, **--requirements-file** `<REQUIREMENTS>`
: A file containing a list of collections to be downloaded.

-s `<API_SERVER>`, **--server** `<API_SERVER>`
: The Galaxy API server URL

collection init

Creates the skeleton framework of a role or collection that complies with the Galaxy metadata format. Requires a role or collection name. The collection name must be in the format `<namespace>.<collection>`.

--collection-skeleton `<COLLECTION_SKELETON>`
: The path to a collection skeleton that the new collection should be based upon.

--init-path `<INIT_PATH>`
: The path in which the skeleton collection will be created. The default is the current working directory.

--timeout `<TIMEOUT>`
: The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-f, --force
: Force overwriting an existing role or collection

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

collection build

Build an Ansible Galaxy collection artifact that can be stored in a central repository like Ansible Galaxy. By default, this command builds from the current working directory. You can optionally pass in the collection input path (where the `galaxy.yml` file is).

--output-path <OUTPUT_PATH>
: The path in which the collection is built to. The default is the current working directory.

--timeout <TIMEOUT>
: The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-f, --force
: Force overwriting an existing role or collection

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

collection publish

Publish a collection into Ansible Galaxy. Requires the path to the collection tarball to publish.

--import-timeout <IMPORT_TIMEOUT>
: The time to wait for the collection import process to finish.

--no-wait
: Don't wait for import validation results.

--timeout <TIMEOUT>
: The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

collection install

--clear-response-cache

Clear the existing server response cache.

--disable-gpg-verify

Disable GPG signature verification when installing collections from a Galaxy server

--force-with-deps

Force overwriting an existing collection and its dependencies.

--ignore-signature-status-code

A status code to ignore during signature verification (for example, NO_PUBKEY). Provide this option multiple times to ignore a list of status codes. Descriptions for the choices can be seen at L(https://github.com/gpg/gnupg/blob/master/doc/DETAILS#general-status-codes).

--keyring `<KEYRING>`

The keyring used during signature verification

--no-cache

Do not use the server response cache.

--offline

Install collection artifacts (tarballs) without contacting any distribution servers. This does not apply to collections in remote Git repositories or URLs to remote tarballs.

--pre

Include pre-release versions. Semantic versioning pre-releases are ignored by default

--required-valid-signature-count `<REQUIRED_VALID_SIGNATURE_COUNT>`

The number of signatures that must successfully verify the collection. This should be a positive integer or -1 to signify that all signatures must be used to verify the collection. Prepend the value with + to fail if no valid signatures are found for the collection (e.g. +all).

--signature

An additional signature source to verify the authenticity of the MANIFEST.json before installing the collection from a Galaxy server. Use in conjunction with a positional collection name (mutually exclusive with –requirements-file).

--timeout `<TIMEOUT>`

The time to wait for operations against the galaxy server, defaults to 60s.

--token `<API_KEY>`, **--api-key** `<API_KEY>`

The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-U, --upgrade

Upgrade installed collection artifacts. This will also update dependencies unless –no-deps is provided

-c, --ignore-certs

Ignore SSL certificate validation errors.

-f, --force

Force overwriting an existing role or collection

-i, --ignore-errors

Ignore errors during installation and continue with the next specified collection. This will not ignore dependency conflict errors.

-n, --no-deps
Don't download collections listed as dependencies.

-p <COLLECTIONS_PATH>, --collections-path <COLLECTIONS_PATH>
The path to the directory containing your collections.

-r <REQUIREMENTS>, --requirements-file <REQUIREMENTS>
A file containing a list of collections to be installed.

-s <API_SERVER>, --server <API_SERVER>
The Galaxy API server URL

collection list

List installed collections or roles

--format <OUTPUT_FORMAT>
Format to display the list of collections in.

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, --api-key <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
Ignore SSL certificate validation errors.

-p, --collections-path
One or more directories to search for collections in addition to the default COLLECTIONS_PATHS. Separate multiple paths with ':'.

-s <API_SERVER>, --server <API_SERVER>
The Galaxy API server URL

collection verify

--ignore-signature-status-code
A status code to ignore during signature verification (for example, NO_PUBKEY). Provide this option multiple times to ignore a list of status codes. Descriptions for the choices can be seen at L(https://github.com/gpg/gnupg/blob/master/doc/DETAILS#general-status-codes).

--keyring <KEYRING>
The keyring used during signature verification

--offline
Validate collection integrity locally without contacting server for canonical manifest hash.

--required-valid-signature-count <REQUIRED_VALID_SIGNATURE_COUNT>
The number of signatures that must successfully verify the collection. This should be a positive integer or all to signify that all signatures must be used to verify the collection. Prepend the value with + to fail if no valid signatures are found for the collection (e.g. +all).

--signature
An additional signature source to verify the authenticity of the MANIFEST.json before using it to verify the rest of the contents of a collection from a Galaxy server. Use in conjunction with a positional collection name (mutually exclusive with –requirements-file).

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, **--ignore-certs**
Ignore SSL certificate validation errors.

-i, **--ignore-errors**
Ignore errors during verification and continue with the next specified collection.

-p, **--collections-path**
One or more directories to search for collections in addition to the default COLLECTIONS_PATHS. Separate multiple paths with ':'.

-r <REQUIREMENTS>, **--requirements-file** <REQUIREMENTS>
A file containing a list of collections to be verified.

-s <API_SERVER>, **--server** <API_SERVER>
The Galaxy API server URL

role

Perform the action on an Ansible Galaxy role. Must be combined with a further action like delete/install/init as listed below.

role init

Creates the skeleton framework of a role or collection that complies with the Galaxy metadata format. Requires a role or collection name. The collection name must be in the format `<namespace>.<collection>`.

--init-path <INIT_PATH>
The path in which the skeleton role will be created. The default is the current working directory.

--offline
Don't query the galaxy API when creating roles

--role-skeleton <ROLE_SKELETON>
The path to a role skeleton that the new role should be based upon.

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

--type <ROLE_TYPE>
Initialize using an alternate role type. Valid types include: 'container', 'apb' and 'network'.

-c, --ignore-certs
Ignore SSL certificate validation errors.

-f, --force
Force overwriting an existing role or collection

-s <API_SERVER>, **--server** <API_SERVER>
The Galaxy API server URL

role remove

removes the list of roles passed as arguments from the local system.

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
Ignore SSL certificate validation errors.

-p, --roles-path
The path to the directory containing your roles. The default is the first writable one configured via DEFAULT_ROLES_PATH: {{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}

-s <API_SERVER>, **--server** <API_SERVER>
The Galaxy API server URL

role delete

Delete a role from Ansible Galaxy.

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
Ignore SSL certificate validation errors.

-s <API_SERVER>, **--server** <API_SERVER>
The Galaxy API server URL

role list

List installed collections or roles

--timeout <TIMEOUT>
: The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-p, --roles-path
: The path to the directory containing your roles. The default is the first writable one configured via DEFAULT_ROLES_PATH: {{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

role search

searches for roles on the Ansible Galaxy server

--author <AUTHOR>
: GitHub username

--galaxy-tags <GALAXY_TAGS>
: list of galaxy tags to filter by

--platforms <PLATFORMS>
: list of OS platforms to filter by

--timeout <TIMEOUT>
: The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

role import

used to import a role into Ansible Galaxy

--branch <REFERENCE>
: The name of a branch to import. Defaults to the repository's default branch (usually master)

--no-wait
: Don't wait for import results.

--role-name <ROLE_NAME>
The name the role should have, if different than the repo name

--status
Check the status of the most recent import request for given github_user/github_repo.

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
Ignore SSL certificate validation errors.

-s <API_SERVER>, **--server** <API_SERVER>
The Galaxy API server URL

role setup

Setup an integration from Github or Travis for Ansible Galaxy roles

--list
List all of your integrations.

--remove <REMOVE_ID>
Remove the integration matching the provided ID value. Use –list to see ID values.

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
Ignore SSL certificate validation errors.

-p, --roles-path
The path to the directory containing your roles. The default is the first writable one configured via DEFAULT_ROLES_PATH: {{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}

-s <API_SERVER>, **--server** <API_SERVER>
The Galaxy API server URL

role info

prints out detailed information about an installed role as well as info available from the galaxy API.

--offline
Don't query the galaxy API when creating roles

--timeout <TIMEOUT>
The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-p, --roles-path
: The path to the directory containing your roles. The default is the first writable one configured via DEFAULT_ROLES_PATH: {{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

role install

--force-with-deps
: Force overwriting an existing role and its dependencies.

--timeout <TIMEOUT>
: The time to wait for operations against the galaxy server, defaults to 60s.

--token <API_KEY>, **--api-key** <API_KEY>
: The Ansible Galaxy API key which can be found at https://galaxy.ansible.com/me/preferences.

-c, --ignore-certs
: Ignore SSL certificate validation errors.

-f, --force
: Force overwriting an existing role or collection

-g, --keep-scm-meta
: Use tar instead of the scm archive option when packaging the role.

-i, --ignore-errors
: Ignore errors and continue with the next specified role.

-n, --no-deps
: Don't download roles listed as dependencies.

-p, --roles-path
: The path to the directory containing your roles. The default is the first writable one configured via DEFAULT_ROLES_PATH: {{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}

-r <REQUIREMENTS>, **--role-file** <REQUIREMENTS>
: A file containing a list of roles to be installed.

-s <API_SERVER>, **--server** <API_SERVER>
: The Galaxy API server URL

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-inventory

None

Synopsis

```
usage: ansible-inventory [-h] [--version] [-v] [-i INVENTORY] [-l SUBSET] [--vault-id␣
↪VAULT_IDS] [--ask-vault-password | --vault-password-file VAULT_PASSWORD_FILES] [--
↪playbook-dir BASEDIR] [-e EXTRA_VARS]
                         [--list] [--host HOST] [--graph] [-y] [--toml] [--vars] [--export]␣
↪[--output OUTPUT_FILE]
                         [host|group]
```

Description

used to display or dump the configured inventory as Ansible sees it

Common Options

--ask-vault-password, --ask-vault-pass
: ask for vault password

--export
: When doing an –list, represent in a way that is optimized for export,not as an accurate representation of how Ansible has processed it

--graph
: create inventory graph, if supplying pattern it must be a valid group name. It will ignore limit

--host <HOST>
: Output specific host info, works as inventory script. It will ignore limit

--list
: Output all hosts info, works as inventory script

--list-hosts
: ==SUPPRESS==

--output <OUTPUT_FILE>
: When doing –list, send the inventory to a file instead of to the screen

--playbook-dir <BASEDIR>
: Since this tool does not use playbooks, use this as a substitute playbook directory. This sets the relative path for many features including roles/ group_vars/ etc.

--toml
: Use TOML format instead of default JSON, ignored for –graph

--vars
: Add vars to graph display, ignored unless used with –graph

--vault-id
: the vault identity to use

--vault-password-file, --vault-pass-file
: vault password file

--version
show program's version number, config file location, configured module search path, module location, executable location and exit

-e, --extra-vars
set additional variables as key=value or YAML/JSON, if filename prepend with @

-h, --help
show this help message and exit

-i, --inventory, --inventory-file
specify inventory host path or comma separated host list. –inventory-file is deprecated

-l <SUBSET>, **--limit** <SUBSET>
further limit selected hosts to an additional pattern

-v, --verbose
Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

-y, --yaml
Use YAML format instead of default JSON, ignored for –graph

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-playbook

Runs Ansible playbooks, executing the defined tasks on the targeted hosts.

Synopsis

```
usage: ansible-playbook [-h] [--version] [-v] [--private-key PRIVATE_KEY_FILE] [-u␣
↪REMOTE_USER] [-c CONNECTION] [-T TIMEOUT] [--ssh-common-args SSH_COMMON_ARGS] [--sftp-
↪extra-args SFTP_EXTRA_ARGS]
                        [--scp-extra-args SCP_EXTRA_ARGS] [--ssh-extra-args SSH_EXTRA_ARGS]␣
↪[-k | --connection-password-file CONNECTION_PASSWORD_FILE] [--force-handlers] [--flush-
↪cache] [-b]
                        [--become-method BECOME_METHOD] [--become-user BECOME_USER] [-K | --
↪become-password-file BECOME_PASSWORD_FILE] [-t TAGS] [--skip-tags SKIP_TAGS] [-C] [-D]␣
↪[-i INVENTORY] [--list-hosts]
                        [-l SUBSET] [-e EXTRA_VARS] [--vault-id VAULT_IDS] [--ask-vault-
↪password | --vault-password-file VAULT_PASSWORD_FILES] [-f FORKS] [-M MODULE_PATH] [--
↪syntax-check] [--list-tasks]
                        [--list-tags] [--step] [--start-at-task START_AT_TASK]
                        playbook [playbook ...]
```

Description

the tool to run *Ansible playbooks*, which are a configuration and multinode deployment system. See the project home page (https://docs.ansible.com) for more information.

Common Options

--ask-vault-password, **--ask-vault-pass**
ask for vault password

--become-method <BECOME_METHOD>
privilege escalation method to use (default=sudo), use *ansible-doc -t become -l* to list valid choices.

--become-password-file <BECOME_PASSWORD_FILE>, **--become-pass-file** <BECOME_PASSWORD_FILE>
Become password file

--become-user <BECOME_USER>
run operations as this user (default=root)

--connection-password-file <CONNECTION_PASSWORD_FILE>,
--conn-pass-file <CONNECTION_PASSWORD_FILE>
Connection password file

--flush-cache
clear the fact cache for every host in inventory

--force-handlers
run handlers even if a task fails

--list-hosts
outputs a list of matching hosts; does not execute anything else

--list-tags
list all available tags

--list-tasks
list all tasks that would be executed

--private-key <PRIVATE_KEY_FILE>, **--key-file** <PRIVATE_KEY_FILE>
use this file to authenticate the connection

--scp-extra-args <SCP_EXTRA_ARGS>
specify extra arguments to pass to scp only (e.g. -l)

--sftp-extra-args <SFTP_EXTRA_ARGS>
specify extra arguments to pass to sftp only (e.g. -f, -l)

--skip-tags
only run plays and tasks whose tags do not match these values

--ssh-common-args <SSH_COMMON_ARGS>
specify common arguments to pass to sftp/scp/ssh (e.g. ProxyCommand)

--ssh-extra-args <SSH_EXTRA_ARGS>
specify extra arguments to pass to ssh only (e.g. -R)

--start-at-task <START_AT_TASK>
start the playbook at the task matching this name

--step
one-step-at-a-time: confirm each task before running

--syntax-check
perform a syntax check on the playbook, but do not execute it

--vault-id
the vault identity to use

--vault-password-file, --vault-pass-file
vault password file

--version
show program's version number, config file location, configured module search path, module location, executable location and exit

-C, --check
don't make any changes; instead, try to predict some of the changes that may occur

-D, --diff
when changing (small) files and templates, show the differences in those files; works great with –check

-K, --ask-become-pass
ask for privilege escalation password

-M, --module-path
prepend colon-separated path(s) to module library (default={{ ANSIBLE_HOME ~ "/plugins/modules:/usr/share/ansible/plugins/modules" }})

-T <TIMEOUT>, **--timeout** <TIMEOUT>
override the connection timeout in seconds (default=10)

-b, --become
run operations with become (does not imply password prompting)

-c <CONNECTION>, **--connection** <CONNECTION>
connection type to use (default=smart)

-e, --extra-vars
set additional variables as key=value or YAML/JSON, if filename prepend with @

-f <FORKS>, **--forks** <FORKS>
specify number of parallel processes to use (default=5)

-h, --help
show this help message and exit

-i, --inventory, --inventory-file
specify inventory host path or comma separated host list. –inventory-file is deprecated

-k, --ask-pass
ask for connection password

-l <SUBSET>, **--limit** <SUBSET>
further limit selected hosts to an additional pattern

-t, --tags
only run plays and tasks tagged with these values

-u <REMOTE_USER>, **--user** <REMOTE_USER>
connect as this user (default=None)

-v, --verbose
Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-pull

pulls playbooks from a VCS repo and executes them for the local host

Synopsis

```
usage: ansible-pull [-h] [--version] [-v] [--private-key PRIVATE_KEY_FILE] [-u REMOTE_
↪USER] [-c CONNECTION] [-T TIMEOUT] [--ssh-common-args SSH_COMMON_ARGS] [--sftp-extra-
↪args SFTP_EXTRA_ARGS]
                 [--scp-extra-args SCP_EXTRA_ARGS] [--ssh-extra-args SSH_EXTRA_ARGS] [-k␣
↪| --connection-password-file CONNECTION_PASSWORD_FILE] [--vault-id VAULT_IDS]
                 [--ask-vault-password | --vault-password-file VAULT_PASSWORD_FILES] [-e␣
↪EXTRA_VARS] [-t TAGS] [--skip-tags SKIP_TAGS] [-i INVENTORY] [--list-hosts] [-l␣
↪SUBSET] [-M MODULE_PATH]
                 [-K | --become-password-file BECOME_PASSWORD_FILE] [--purge] [-o] [-s␣
↪SLEEP] [-f] [-d DEST] [-U URL] [--full] [-C CHECKOUT] [--accept-host-key] [-m MODULE_
↪NAME] [--verify-commit] [--clean]
                 [--track-subs] [--check] [--diff]
                 [playbook.yml ...]
```

Description

Used to pull a remote copy of ansible on each managed node, each set to run via cron and update playbook source via a source repository. This inverts the default *push* architecture of ansible into a *pull* architecture, which has near-limitless scaling potential.

None of the CLI tools are designed to run concurrently with themselves, you should use an external scheduler and/or locking to ensure there are no clashing operations.

The setup playbook can be tuned to change the cron frequency, logging locations, and parameters to ansible-pull. This is useful both for extreme scale-out as well as periodic remediation. Usage of the 'fetch' module to retrieve logs from ansible-pull runs would be an excellent way to gather and analyze remote logs from ansible-pull.

Common Options

`--accept-host-key`

adds the hostkey for the repo url if not already added

`--ask-vault-password`, **`--ask-vault-pass`**

ask for vault password

`--become-password-file` `<BECOME_PASSWORD_FILE>`, **`--become-pass-file`** `<BECOME_PASSWORD_FILE>`

Become password file

`--check`

don't make any changes; instead, try to predict some of the changes that may occur

`--clean`

modified files in the working repository will be discarded

`--connection-password-file` `<CONNECTION_PASSWORD_FILE>`, **`--conn-pass-file`** `<CONNECTION_PASSWORD_FILE>`

Connection password file

`--diff`

when changing (small) files and templates, show the differences in those files; works great with –check

`--full`
: Do a full clone, instead of a shallow one.

`--list-hosts`
: outputs a list of matching hosts; does not execute anything else

`--private-key` `<PRIVATE_KEY_FILE>`, **`--key-file`** `<PRIVATE_KEY_FILE>`
: use this file to authenticate the connection

`--purge`
: purge checkout after playbook run

`--scp-extra-args` `<SCP_EXTRA_ARGS>`
: specify extra arguments to pass to scp only (e.g. -l)

`--sftp-extra-args` `<SFTP_EXTRA_ARGS>`
: specify extra arguments to pass to sftp only (e.g. -f, -l)

`--skip-tags`
: only run plays and tasks whose tags do not match these values

`--ssh-common-args` `<SSH_COMMON_ARGS>`
: specify common arguments to pass to sftp/scp/ssh (e.g. ProxyCommand)

`--ssh-extra-args` `<SSH_EXTRA_ARGS>`
: specify extra arguments to pass to ssh only (e.g. -R)

`--track-subs`
: submodules will track the latest changes. This is equivalent to specifying the –remote flag to git submodule update

`--vault-id`
: the vault identity to use

`--vault-password-file`, **`--vault-pass-file`**
: vault password file

`--verify-commit`
: verify GPG signature of checked out commit, if it fails abort running the playbook. This needs the corresponding VCS module to support such an operation

`--version`
: show program's version number, config file location, configured module search path, module location, executable location and exit

`-C` `<CHECKOUT>`, **`--checkout`** `<CHECKOUT>`
: branch/tag/commit to checkout. Defaults to behavior of repository module.

`-K`, **`--ask-become-pass`**
: ask for privilege escalation password

`-M`, **`--module-path`**
: prepend colon-separated path(s) to module library (default={{ ANSIBLE_HOME ~ "/plugins/modules:/usr/share/ansible/plugins/modules" }})

`-T` `<TIMEOUT>`, **`--timeout`** `<TIMEOUT>`
: override the connection timeout in seconds (default=10)

-U <URL>, **--url** <URL>
URL of the playbook repository

-c <CONNECTION>, **--connection** <CONNECTION>
connection type to use (default=smart)

-d <DEST>, **--directory** <DEST>
absolute path of repository checkout directory (relative paths are not supported)

-e, **--extra-vars**
set additional variables as key=value or YAML/JSON, if filename prepend with @

-f, **--force**
run the playbook even if the repository could not be updated

-h, **--help**
show this help message and exit

-i, **--inventory**, **--inventory-file**
specify inventory host path or comma separated host list. –inventory-file is deprecated

-k, **--ask-pass**
ask for connection password

-l <SUBSET>, **--limit** <SUBSET>
further limit selected hosts to an additional pattern

-m <MODULE_NAME>, **--module-name** <MODULE_NAME>
Repository module name, which ansible will use to check out the repo. Choices are ('git', 'subversion', 'hg', 'bzr'). Default is git.

-o, **--only-if-changed**
only run the playbook if the repository has been updated

-s <SLEEP>, **--sleep** <SLEEP>
sleep for random interval (between 0 and n number of seconds) before starting. This is a useful way to disperse git requests

-t, **--tags**
only run plays and tasks tagged with these values

-u <REMOTE_USER>, **--user** <REMOTE_USER>
connect as this user (default=None)

-v, **--verbose**
Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Environment

The following environment variables may be specified.

ANSIBLE_CONFIG – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

ansible-vault

encryption/decryption utility for Ansible data files

Synopsis

```
usage: ansible-vault [-h] [--version] [-v] {create,decrypt,edit,view,encrypt,encrypt_
↪string,rekey} ...
```

Description

can encrypt any structured data file used by Ansible. This can include *group_vars/* or *host_vars/* inventory variables, variables loaded by *include_vars* or *vars_files*, or variable files passed on the ansible-playbook command line with *-e @file.yml* or *-e @file.json*. Role variables and defaults are also included!

Because Ansible tasks, handlers, and other objects are data, these can also be encrypted with vault. If you'd like to not expose what variables you are using, you can keep an individual task file entirely encrypted.

Common Options

`--version`
: show program's version number, config file location, configured module search path, module location, executable location and exit

`-h, --help`
: show this help message and exit

`-v, --verbose`
: Causes Ansible to print more debug messages. Adding multiple -v will increase the verbosity, the builtin plugins currently evaluate up to -vvvvvv. A reasonable level to start is -vvv, connection debugging might require -vvvv.

Actions

create

create and open a file in an editor that will be encrypted with the provided vault secret when closed

`--ask-vault-password, --ask-vault-pass`
: ask for vault password

`--encrypt-vault-id` `<ENCRYPT_VAULT_ID>`
: the vault id used to encrypt (required if more than one vault-id is provided)

`--vault-id`
: the vault identity to use

`--vault-password-file, --vault-pass-file`
vault password file

decrypt

decrypt the supplied file using the provided vault secret

`--ask-vault-password, --ask-vault-pass`
ask for vault password

`--output` `<OUTPUT_FILE>`
output file name for encrypt or decrypt; use - for stdout

`--vault-id`
the vault identity to use

`--vault-password-file, --vault-pass-file`
vault password file

edit

open and decrypt an existing vaulted file in an editor, that will be encrypted again when closed

`--ask-vault-password, --ask-vault-pass`
ask for vault password

`--encrypt-vault-id` `<ENCRYPT_VAULT_ID>`
the vault id used to encrypt (required if more than one vault-id is provided)

`--vault-id`
the vault identity to use

`--vault-password-file, --vault-pass-file`
vault password file

view

open, decrypt and view an existing vaulted file using a pager using the supplied vault secret

`--ask-vault-password, --ask-vault-pass`
ask for vault password

`--vault-id`
the vault identity to use

`--vault-password-file, --vault-pass-file`
vault password file

encrypt

encrypt the supplied file using the provided vault secret

--ask-vault-password, --ask-vault-pass
  ask for vault password

--encrypt-vault-id <ENCRYPT_VAULT_ID>
  the vault id used to encrypt (required if more than one vault-id is provided)

--output <OUTPUT_FILE>
  output file name for encrypt or decrypt; use - for stdout

--vault-id
  the vault identity to use

--vault-password-file, --vault-pass-file
  vault password file

encrypt_string

encrypt the supplied string using the provided vault secret

--ask-vault-password, --ask-vault-pass
  ask for vault password

--encrypt-vault-id <ENCRYPT_VAULT_ID>
  the vault id used to encrypt (required if more than one vault-id is provided)

--output <OUTPUT_FILE>
  output file name for encrypt or decrypt; use - for stdout

--show-input
  Do not hide input when prompted for the string to encrypt

--stdin-name <ENCRYPT_STRING_STDIN_NAME>
  Specify the variable name for stdin

--vault-id
  the vault identity to use

--vault-password-file, --vault-pass-file
  vault password file

-n, --name
  Specify the variable name

-p, --prompt
  Prompt for the string to encrypt

rekey

re-encrypt a vaulted file with a new secret, the previous secret is required

`--ask-vault-password`, `--ask-vault-pass`
ask for vault password

`--encrypt-vault-id` `<ENCRYPT_VAULT_ID>`
the vault id used to encrypt (required if more than one vault-id is provided)

`--new-vault-id` `<NEW_VAULT_ID>`
the new vault identity to use for rekey

`--new-vault-password-file` `<NEW_VAULT_PASSWORD_FILE>`
new vault password file for rekey

`--vault-id`
the vault identity to use

`--vault-password-file`, `--vault-pass-file`
vault password file

Environment

The following environment variables may be specified.

`ANSIBLE_CONFIG` – Override the default ansible config file

Many more are available for most options in ansible.cfg

Files

`/etc/ansible/ansible.cfg` – Config file, used if present

`~/.ansible.cfg` – User config file, overrides the default config if present

Author

Ansible was originally written by Michael DeHaan.

See the *AUTHORS* file for a complete list of contributors.

License

Ansible is released under the terms of the GPLv3+ License.

See also

ansible(1), *ansible-config(1)*, *ansible-console(1)*, *ansible-doc(1)*, *ansible-galaxy(1)*, *ansible-inventory(1)*, *ansible-playbook(1)*, *ansible-pull(1)*, *ansible-vault(1)*,

1.5.3 Ansible CLI cheatsheet

This page shows one or more examples of each Ansible command line utility with some common flags added and a link to the full documentation for the command. This page offers a quick reminder of some common use cases only - it may be out of date or incomplete or both. For canonical documentation, follow the links to the CLI pages.

ansible-playbook

```
ansible-playbook -i /path/to/my_inventory_file -u my_connection_user -k -f 3 -T 30 -t my_
↪tag -m /path/to/my_modules -b -K my_playbook.yml
```

Loads `my_playbook.yml` from the current working directory and:

- `-i` - uses `my_inventory_file` in the path provided for *inventory* to match the *pattern*.
- `-u` - connects *over SSH* as `my_connection_user`.
- `-k` - asks for password which is then provided to SSH authentication.
- `-f` - allocates 3 *forks*.
- `-T` - sets a 30-second timeout.
- `-t` - runs only tasks marked with the *tag* `my_tag`.
- `-m` - loads *local modules* from `/path/to/my/modules`.
- `-b` - executes with elevated privileges (uses *become*).
- `-K` - prompts the user for the become password.

See *ansible-playbook* for detailed documentation.

1.6 Using Ansible playbooks

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible playbooks guide. Playbooks are automation blueprints, in `YAML` format, that Ansible uses to deploy and configure nodes in an inventory. This guide introduces you to playbooks and then covers different use cases for tasks and plays, such as:

- Executing tasks with elevated privileges or as a different user.

- Using loops to repeat tasks for items in a list.
- Delegating playbooks to execute tasks on different machines.
- Running conditional tasks and evaluating conditions with playbook tests.
- Using blocks to group sets of tasks.

You can also learn how to use Ansible playbooks more effectively by creating re-usable files and roles, including and importing playbooks, and running selected parts of a playbook with tags.

1.6.1 Ansible playbooks

Ansible Playbooks offer a repeatable, re-usable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications. If you need to execute a task with Ansible more than once, write a playbook and put it under source control. Then you can use the playbook to push out new configuration or confirm the configuration of remote systems. The playbooks in the ansible-examples repository illustrate many useful techniques. You may want to look at these in another tab as you read the documentation.

Playbooks can:

- declare configurations
- orchestrate steps of any manual ordered process, on multiple sets of machines, in a defined order
- launch tasks synchronously or *asynchronously*

Playbook syntax

Playbooks are expressed in YAML format with a minimum of syntax. If you are not familiar with YAML, look at our overview of *YAML Syntax* and consider installing an add-on for your text editor (see *Other Tools and Programs*) to help you write clean YAML syntax in your playbooks.

A playbook is composed of one or more 'plays' in an ordered list. The terms 'playbook' and 'play' are sports analogies. Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module.

Playbook execution

A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple 'plays' can orchestrate multi-machine deployments, running one play on your webservers, then another play on your database servers, then a third play on your network infrastructure, and so on. At a minimum, each play defines two things:

- the managed nodes to target, using a *pattern*
- at least one task to execute

Note: In Ansible 2.10 and later, we recommend you use the fully-qualified collection name in your playbooks to ensure the correct module is selected, because multiple collections can contain modules with the same name (for example, `user`). See *Using collections in a playbook*.

In this example, the first play targets the web servers; the second play targets the database servers.

```yaml
---
- name: Update web servers
  hosts: webservers
  remote_user: root

  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- name: Update db servers
  hosts: databases
  remote_user: root

  tasks:
  - name: Ensure postgresql is at the latest version
    ansible.builtin.yum:
      name: postgresql
      state: latest
  - name: Ensure that postgresql is started
    ansible.builtin.service:
      name: postgresql
      state: started
```

Your playbook can include more than just a hosts line and tasks. For example, the playbook above sets a `remote_user` for each play. This is the user account for the SSH connection. You can add other *Playbook Keywords* at the playbook, play, or task level to influence how Ansible behaves. Playbook keywords can control the *connection plugin*, whether to use *privilege escalation*, how to handle errors, and more. To support a variety of environments, Ansible lets you set many of these parameters as command-line flags, in your Ansible configuration, or in your inventory. Learning the *precedence rules* for these sources of data will help you as you expand your Ansible ecosystem.

Task execution

By default, Ansible executes each task in order, one at a time, against all machines matched by the host pattern. Each task executes a module with specific arguments. When a task has executed on all target machines, Ansible moves on to the next task. You can use *strategies* to change this default behavior. Within each play, Ansible applies the same task directives to all hosts. If a task fails on a host, Ansible takes that host out of the rotation for the rest of the playbook.

When you run a playbook, Ansible returns information about connections, the `name` lines of all your plays and tasks, whether each task has succeeded or failed on each machine, and whether each task has made a change on each machine. At the bottom of the playbook execution, Ansible provides a summary of the nodes that were targeted and how they performed. General failures and fatal "unreachable" communication attempts are kept separate in the counts.

Desired state and 'idempotency'

Most Ansible modules check whether the desired final state has already been achieved, and exit without performing any actions if that state has been achieved, so that repeating the task does not change the final state. Modules that behave this way are often called 'idempotent.' Whether you run a playbook once, or multiple times, the outcome should be the same. However, not all playbooks and not all modules behave this way. If you are unsure, test your playbooks in a sandbox environment before running them multiple times in production.

Running playbooks

To run your playbook, use the *ansible-playbook* command.

```
ansible-playbook playbook.yml -f 10
```

Use the `--verbose` flag when running your playbook to see detailed output from successful modules as well as unsuccessful ones.

Ansible-Pull

Should you want to invert the architecture of Ansible, so that nodes check in to a central location, instead of pushing configuration out to them, you can.

The `ansible-pull` is a small script that will checkout a repo of configuration instructions from git, and then run `ansible-playbook` against that content.

Assuming you load balance your checkout location, `ansible-pull` scales essentially infinitely.

Run `ansible-pull --help` for details.

There's also a clever playbook available to configure `ansible-pull` through a crontab from push mode.

Verifying playbooks

You may want to verify your playbooks to catch syntax errors and other problems before you run them. The *ansible-playbook* command offers several options for verification, including `--check`, `--diff`, `--list-hosts`, `--list-tasks`, and `--syntax-check`. The *Tools for validating playbooks* describes other tools for validating and testing playbooks.

ansible-lint

You can use ansible-lint for detailed, Ansible-specific feedback on your playbooks before you execute them. For example, if you run `ansible-lint` on the playbook called `verify-apache.yml` near the top of this page, you should get the following results:

```
$ ansible-lint verify-apache.yml
[403] Package installs should not use latest
verify-apache.yml:8
Task/Handler: ensure apache is at the latest version
```

The ansible-lint default rules page describes each error. For `[403]`, the recommended fix is to change `state: latest` to `state:  present` in the playbook.

See also:

ansible-lint
  Learn how to test Ansible Playbooks syntax

YAML Syntax
  Learn about YAML syntax

General tips
  Tips for managing playbooks in the real world

Collection Index
  Browse existing collections, modules, and plugins

Should you develop a module?
  Learn to extend Ansible by writing your own modules

Patterns: targeting hosts and groups
  Learn about how to select hosts

GitHub examples directory
  Complete end-to-end playbook examples

Mailing List
  Questions? Help? Ideas? Stop by the list on Google Groups

1.6.2 Working with playbooks

Playbooks record and execute Ansible's configuration, deployment, and orchestration functions. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.

If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.

At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

Playbooks are designed to be human-readable and are developed in a basic text language. There are multiple ways to organize playbooks and the files they include, and we'll offer up some suggestions on that and making the most out of Ansible.

You should look at Example Playbooks while reading along with the playbook documentation. These illustrate best practices as well as how to put many of the various concepts together.

Templating (Jinja2)

Ansible uses Jinja2 templating to enable dynamic expressions and access to *variables* and *facts*. You can use templating with the template module. For example, you can create a template for a configuration file, then deploy that configuration file to multiple environments and supply the correct data (IP address, hostname, version) for each environment. You can also use templating in playbooks directly, by templating task names and more. You can use all the standard filters and tests included in Jinja2. Ansible includes additional specialized filters for selecting and transforming data, tests for evaluating template expressions, and *Lookup plugins* for retrieving data from external sources such as files, APIs, and databases for use in templating.

All templating happens on the Ansible controller **before** the task is sent and executed on the target machine. This approach minimizes the package requirements on the target (jinja2 is only required on the controller). It also limits the amount of data Ansible passes to the target machine. Ansible parses templates on the controller and passes only the information needed for each task to the target machine, instead of passing all the data on the controller and parsing it on the target.

Note: Files and data used by the template module must be utf-8 encoded.

Jinja2 Example

In this example, we want to write the server hostname to its /tmp/hostname.

Our directory looks like this:

```
├── hostname.yml
├── templates
    └── test.j2
```

Our hostname.yml:

```
---
- name: Write hostname
  hosts: all
  tasks:
  - name: write hostname using jinja2
    ansible.builtin.template:
       src: templates/test.j2
       dest: /tmp/hostname
```

Our test.j2:

```
My name is {{ ansible_facts['hostname'] }}
```

See also:

Ansible playbooks
An introduction to playbooks

Playbook tips
Tips and tricks for playbooks

Jinja2 Docs
Jinja2 documentation, includes the syntax and semantics of the templates

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Using filters to manipulate data

Filters let you transform JSON data into YAML data, split a URL to extract the hostname, get the SHA1 hash of a string, add or multiply integers, and much more. You can use the Ansible-specific filters documented here to manipulate your data, or use any of the standard filters shipped with Jinja2 - see the list of built-in filters in the official Jinja2 template documentation. You can also use Python methods to transform data. You can *create custom Ansible filters as plugins*, though we generally welcome new filters into the ansible-core repo so everyone can use them.

Because templating happens on the Ansible controller, **not** on the target host, filters execute on the controller and transform data locally.

Handling undefined variables

Filters can help you manage missing or undefined variables by providing defaults or making some variables optional. If you configure Ansible to ignore most undefined variables, you can mark some variables as requiring values with the `mandatory` filter.

Providing default values

You can provide default values for variables directly in your templates using the Jinja2 'default' filter. This is often a better approach than failing if a variable is not defined:

```
{{ some_variable | default(5) }}
```

In the above example, if the variable 'some_variable' is not defined, Ansible uses the default value 5, rather than raising an "undefined variable" error and failing. If you are working within a role, you can also add a `defaults/main.yml` to define the default values for variables in your role.

Beginning in version 2.8, attempting to access an attribute of an Undefined value in Jinja will return another Undefined value, rather than throwing an error immediately. This means that you can now simply use a default with a value in a nested data structure (in other words, `{{ foo.bar.baz | default('DEFAULT') }}`) when you do not know if the intermediate values are defined.

If you want to use the default value when variables evaluate to false or an empty string you have to set the second parameter to `true`:

```
{{ lookup('env', 'MY_USER') | default('admin', true) }}
```

Making variables optional

By default Ansible requires values for all variables in a templated expression. However, you can make specific variables optional. For example, you might want to use a system default for some items and control the value for others. To make a variable optional, set the default value to the special variable `omit`:

```
- name: Touch files with an optional mode
  ansible.builtin.file:
    dest: "{{ item.path }}"
    state: touch
    mode: "{{ item.mode | default(omit) }}"
  loop:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
      mode: "0444"
```

In this example, the default mode for the files `/tmp/foo` and `/tmp/bar` is determined by the umask of the system. Ansible does not send a value for `mode`. Only the third file, `/tmp/baz`, receives the *mode=0444* option.

Note: If you are "chaining" additional filters after the `default(omit)` filter, you should instead do something like this: `"{{ foo | default(None) | some_filter or omit }}"`. In this example, the default `None` (Python null) value will cause the later filters to fail, which will trigger the `or omit` portion of the logic. Using `omit` in this manner is very specific to the later filters you are chaining though, so be prepared for some trial and error if you do this.

Defining mandatory values

If you configure Ansible to ignore undefined variables, you may want to define some values as mandatory. By default, Ansible fails if a variable in your playbook or command is undefined. You can configure Ansible to allow undefined variables by setting *DEFAULT_UNDEFINED_VAR_BEHAVIOR* to `false`. In that case, you may want to require some variables to be defined. You can do this with:

```
{{ variable | mandatory }}
```

The variable value will be used as is, but the template evaluation will raise an error if it is undefined.

A convenient way of requiring a variable to be overridden is to give it an undefined value using the `undef` keyword. This can be useful in a role's defaults.

```
galaxy_url: "https://galaxy.ansible.com"
galaxy_api_key: "{{ undef(hint='You must specify your Galaxy API key') }}"
```

Defining different values for true/false/null (ternary)

You can create a test, then define one value to use when the test returns true and another when the test returns false (new in version 1.9):

```
{{ (status == 'needs_restart') | ternary('restart', 'continue') }}
```

In addition, you can define a one value to use on true, one value on false and a third value on null (new in version 2.8):

```
{{ enabled | ternary('no shutdown', 'shutdown', omit) }}
```

Managing data types

You might need to know, change, or set the data type on a variable. For example, a registered variable might contain a dictionary when your next task needs a list, or a user *prompt* might return a string when your playbook needs a boolean value. Use the `type_debug`, `dict2items`, and `items2dict` filters to manage data types. You can also use the data type itself to cast a value as a specific data type.

Discovering the data type

New in version 2.3.

If you are unsure of the underlying Python type of a variable, you can use the `type_debug` filter to display it. This is useful in debugging when you need a particular type of variable:

```
{{ myvar | type_debug }}
```

You should note that, while this may seem like a useful filter for checking that you have the right type of data in a variable, you should often prefer *type tests*, which will allow you to test for specific data types.

Transforming dictionaries into lists

New in version 2.6.

Use the `dict2items` filter to transform a dictionary into a list of items suitable for *looping*:

```
{{ dict | dict2items }}
```

Dictionary data (before applying the `dict2items` filter):

```
tags:
  Application: payment
  Environment: dev
```

List data (after applying the `dict2items` filter):

```
- key: Application
  value: payment
- key: Environment
  value: dev
```

New in version 2.8.

The dict2items filter is the reverse of the items2dict filter.

If you want to configure the names of the keys, the dict2items filter accepts 2 keyword arguments. Pass the key_name and value_name arguments to configure the names of the keys in the list output:

```
{{ files | dict2items(key_name='file', value_name='path') }}
```

Dictionary data (before applying the dict2items filter):

```
files:
  users: /etc/passwd
  groups: /etc/group
```

List data (after applying the dict2items filter):

```
- file: users
  path: /etc/passwd
- file: groups
  path: /etc/group
```

Transforming lists into dictionaries

New in version 2.7.

Use the items2dict filter to transform a list into a dictionary, mapping the content into key: value pairs:

```
{{ tags | items2dict }}
```

List data (before applying the items2dict filter):

```
tags:
  - key: Application
    value: payment
  - key: Environment
    value: dev
```

Dictionary data (after applying the items2dict filter):

```
Application: payment
Environment: dev
```

The items2dict filter is the reverse of the dict2items filter.

Not all lists use key to designate keys and value to designate values. For example:

```
fruits:
  - fruit: apple
    color: red
  - fruit: pear
    color: yellow
  - fruit: grapefruit
    color: yellow
```

In this example, you must pass the `key_name` and `value_name` arguments to configure the transformation. For example:

```
{{ tags | items2dict(key_name='fruit', value_name='color') }}
```

If you do not pass these arguments, or do not pass the correct values for your list, you will see `KeyError: key` or `KeyError: my_typo`.

Forcing the data type

You can cast values as certain types. For example, if you expect the input "True" from a *vars_prompt* and you want Ansible to recognize it as a boolean value instead of a string:

```
- ansible.builtin.debug:
    msg: test
  when: some_string_value | bool
```

If you want to perform a mathematical comparison on a fact and you want Ansible to recognize it as an integer instead of a string:

```
- shell: echo "only on Red Hat 6, derivatives, and later"
  when: ansible_facts['os_family'] == "RedHat" and ansible_facts['lsb']['major_release']␣
↪| int >= 6
```

New in version 1.6.

Formatting data: YAML and JSON

You can switch a data structure in a template from or to JSON or YAML format, with options for formatting, indenting, and loading data. The basic filters are occasionally useful for debugging:

```
{{ some_variable | to_json }}
{{ some_variable | to_yaml }}
```

For human readable output, you can use:

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

You can change the indentation of either format:

```
{{ some_variable | to_nice_json(indent=2) }}
{{ some_variable | to_nice_yaml(indent=8) }}
```

The `to_yaml` and `to_nice_yaml` filters use the PyYAML library which has a default 80 symbol string length limit. That causes unexpected line break after 80th symbol (if there is a space after 80th symbol) To avoid such behavior and generate long lines, use the `width` option. You must use a hardcoded number to define the width, instead of a construction like `float("inf")`, because the filter does not support proxying Python functions. For example:

```
{{ some_variable | to_yaml(indent=8, width=1337) }}
{{ some_variable | to_nice_yaml(indent=8, width=1337) }}
```

The filter does support passing through other YAML parameters. For a full list, see the PyYAML documentation for `dump()`.

If you are reading in some already formatted data:

```
{{ some_variable | from_json }}
{{ some_variable | from_yaml }}
```

for example:

```
tasks:
  - name: Register JSON output as a variable
    ansible.builtin.shell: cat /some/path/to/file.json
    register: result

  - name: Set a variable
    ansible.builtin.set_fact:
      myvar: "{{ result.stdout | from_json }}"
```

Filter *to_json* and Unicode support

By default *to_json* and *to_nice_json* will convert data received to ASCII, so:

```
{{ 'München'| to_json }}
```

will return:

```
'M\u00fcnchen'
```

To keep Unicode characters, pass the parameter *ensure_ascii=False* to the filter:

```
{{ 'München'| to_json(ensure_ascii=False) }}

'München'
```

New in version 2.7.

To parse multi-document YAML strings, the `from_yaml_all` filter is provided. The `from_yaml_all` filter will return a generator of parsed YAML documents.

for example:

```
tasks:
  - name: Register a file content as a variable
    ansible.builtin.shell: cat /some/path/to/multidoc-file.yaml
    register: result

  - name: Print the transformed variable
    ansible.builtin.debug:
      msg: '{{ item }}'
    loop: '{{ result.stdout | from_yaml_all | list }}'
```

Combining and selecting data

You can combine data from multiple sources and types, and select values from large data structures, giving you precise control over complex data.

Combining items from multiple lists: zip and zip_longest

New in version 2.3.

To get a list combining the elements of other lists use `zip`:

```
- name: Give me list combo of two lists
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5,6] | zip(['a','b','c','d','e','f']) | list }}"

# => [[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, "e"], [6, "f"]]

- name: Give me shortest combo of two lists
  ansible.builtin.debug:
    msg: "{{ [1,2,3] | zip(['a','b','c','d','e','f']) | list }}"

# => [[1, "a"], [2, "b"], [3, "c"]]
```

To always exhaust all lists use `zip_longest`:

```
- name: Give me longest combo of three lists , fill with X
  ansible.builtin.debug:
    msg: "{{ [1,2,3] | zip_longest(['a','b','c','d','e','f'], [21, 22, 23], fillvalue='X
↪') | list }}"

# => [[1, "a", 21], [2, "b", 22], [3, "c", 23], ["X", "d", "X"], ["X", "e", "X"], ["X",
↪"f", "X"]]
```

Similarly to the output of the `items2dict` filter mentioned above, these filters can be used to construct a `dict`:

```
{{ dict(keys_list | zip(values_list)) }}
```

List data (before applying the `zip` filter):

```
keys_list:
  - one
  - two
values_list:
  - apple
  - orange
```

Dictionary data (after applying the `zip` filter):

```
one: apple
two: orange
```

Combining objects and subelements

New in version 2.7.

The subelements filter produces a product of an object and the subelement values of that object, similar to the subelements lookup. This lets you specify individual subelements to use in a template. For example, this expression:

```
{{ users | subelements('groups', skip_missing=True) }}
```

Data before applying the subelements filter:

```
users:
- name: alice
  authorized:
  - /tmp/alice/onekey.pub
  - /tmp/alice/twokey.pub
  groups:
  - wheel
  - docker
- name: bob
  authorized:
  - /tmp/bob/id_rsa.pub
  groups:
  - docker
```

Data after applying the subelements filter:

```
-
  - name: alice
    groups:
    - wheel
    - docker
    authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
  - wheel
-
  - name: alice
    groups:
    - wheel
    - docker
    authorized:
    - /tmp/alice/onekey.pub
    - /tmp/alice/twokey.pub
  - docker
-
  - name: bob
    authorized:
    - /tmp/bob/id_rsa.pub
    groups:
    - docker
  - docker
```

You can use the transformed data with loop to iterate over the same subelement for multiple objects:

```
- name: Set authorized ssh key, extracting just that data from 'users'
  ansible.posix.authorized_key:
    user: "{{ item.0.name }}"
    key: "{{ lookup('file', item.1) }}"
  loop: "{{ users | subelements('authorized') }}"
```

Combining hashes/dictionaries

New in version 2.0.

The `combine` filter allows hashes to be merged. For example, the following would override keys in one hash:

```
{{ {'a':1, 'b':2} | combine({'b':3}) }}
```

The resulting hash would be:

```
{'a':1, 'b':3}
```

The filter can also take multiple arguments to merge:

```
{{ a | combine(b, c, d) }}
{{ [a, b, c, d] | combine }}
```

In this case, keys in `d` would override those in `c`, which would override those in `b`, and so on.

The filter also accepts two optional parameters: `recursive` and `list_merge`.

recursive
Is a boolean, default to `False`. Should the `combine` recursively merge nested hashes. Note: It does **not** depend on the value of the `hash_behaviour` setting in `ansible.cfg`.

list_merge
Is a string, its possible values are `replace` (default), `keep`, `append`, `prepend`, `append_rp` or `prepend_rp`. It modifies the behaviour of `combine` when the hashes to merge contain arrays/lists.

```
default:
  a:
    x: default
    y: default
  b: default
  c: default
patch:
  a:
    y: patch
    z: patch
  b: patch
```

If `recursive=False` (the default), nested hash aren't merged:

```
{{ default | combine(patch) }}
```

This would result in:

```
a:
  y: patch
  z: patch
b: patch
c: default
```

If `recursive=True`, recurse into nested hash and merge their keys:

```
{{ default | combine(patch, recursive=True) }}
```

This would result in:

```
a:
  x: default
  y: patch
  z: patch
b: patch
c: default
```

If `list_merge='replace'` (the default), arrays from the right hash will "replace" the ones in the left hash:

```
default:
  a:
    - default
patch:
  a:
    - patch
```

```
{{ default | combine(patch) }}
```

This would result in:

```
a:
  - patch
```

If `list_merge='keep'`, arrays from the left hash will be kept:

```
{{ default | combine(patch, list_merge='keep') }}
```

This would result in:

```
a:
  - default
```

If `list_merge='append'`, arrays from the right hash will be appended to the ones in the left hash:

```
{{ default | combine(patch, list_merge='append') }}
```

This would result in:

```
a:
  - default
  - patch
```

If `list_merge='prepend'`, arrays from the right hash will be prepended to the ones in the left hash:

```
{{ default | combine(patch, list_merge='prepend') }}
```

This would result in:

```
a:
  - patch
  - default
```

If list_merge='append_rp', arrays from the right hash will be appended to the ones in the left hash. Elements of arrays in the left hash that are also in the corresponding array of the right hash will be removed ("rp" stands for "remove present"). Duplicate elements that aren't in both hashes are kept:

```
default:
  a:
    - 1
    - 1
    - 2
    - 3
patch:
  a:
    - 3
    - 4
    - 5
    - 5
```

```
{{ default | combine(patch, list_merge='append_rp') }}
```

This would result in:

```
a:
  - 1
  - 1
  - 2
  - 3
  - 4
  - 5
  - 5
```

If list_merge='prepend_rp', the behavior is similar to the one for append_rp, but elements of arrays in the right hash are prepended:

```
{{ default | combine(patch, list_merge='prepend_rp') }}
```

This would result in:

```
a:
  - 3
  - 4
  - 5
  - 5
  - 1
  - 1
  - 2
```

`recursive` and `list_merge` can be used together:

```
default:
  a:
    a':
      x: default_value
      y: default_value
      list:
        - default_value
  b:
    - 1
    - 1
    - 2
    - 3
patch:
  a:
    a':
      y: patch_value
      z: patch_value
      list:
        - patch_value
  b:
    - 3
    - 4
    - 4
    - key: value
```

```
{{ default | combine(patch, recursive=True, list_merge='append_rp') }}
```

This would result in:

```
a:
  a':
    x: default_value
    y: patch_value
    z: patch_value
    list:
      - default_value
      - patch_value
b:
  - 1
  - 1
  - 2
  - 3
  - 4
  - 4
  - key: value
```

Selecting values from arrays or hashtables

New in version 2.1.

The *extract* filter is used to map from a list of indices to a list of values from a container (hash or array):

```
{{ [0,2] | map('extract', ['x','y','z']) | list }}
{{ ['x','y'] | map('extract', {'x': 42, 'y': 31}) | list }}
```

The results of the above expressions would be:

```
['x', 'z']
[42, 31]
```

The filter can take another argument:

```
{{ groups['x'] | map('extract', hostvars, 'ec2_ip_address') | list }}
```

This takes the list of hosts in group 'x', looks them up in *hostvars*, and then looks up the *ec2_ip_address* of the result. The final result is a list of IP addresses for the hosts in group 'x'.

The third argument to the filter can also be a list, for a recursive lookup inside the container:

```
{{ ['a'] | map('extract', b, ['x','y']) | list }}
```

This would return a list containing the value of *b['a']['x']['y']*.

Combining lists

This set of filters returns a list of combined lists.

permutations

To get permutations of a list:

```
- name: Give me largest permutations (order matters)
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5] | ansible.builtin.permutations | list }}"

- name: Give me permutations of sets of three
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5] | ansible.builtin.permutations(3) | list }}"
```

combinations

Combinations always require a set size:

```
- name: Give me combinations for sets of two
  ansible.builtin.debug:
    msg: "{{ [1,2,3,4,5] | ansible.builtin.combinations(2) | list }}"
```

Also see the zip_filter

products

The product filter returns the cartesian product of the input iterables. This is roughly equivalent to nested for-loops in a generator expression.

For example:

```
- name: Generate multiple hostnames
  ansible.builtin.debug:
    msg: "{{ ['foo', 'bar'] | product(['com']) | map('join', '.') | join(',') }}"
```

This would result in:

```
{ "msg": "foo.com,bar.com" }
```

Selecting JSON data: JSON queries

To select a single element or a data subset from a complex data structure in JSON format (for example, Ansible facts), use the `json_query` filter. The `json_query` filter lets you query a complex JSON structure and iterate over it using a loop structure.

Note: This filter has migrated to the community.general collection. Follow the installation instructions to install that collection.

Note: You must manually install the **jmespath** dependency on the Ansible controller before using this filter. This filter is built upon **jmespath**, and you can use the same syntax. For examples, see jmespath examples.

Consider this data structure:

```
{
    "domain_definition": {
        "domain": {
            "cluster": [
                {
                    "name": "cluster1"
                },
                {
                    "name": "cluster2"
                }
            ],
            "server": [
                {
                    "name": "server11",
                    "cluster": "cluster1",
                    "port": "8080"
                },
                {
                    "name": "server12",
                    "cluster": "cluster1",
                    "port": "8090"
```

(continues on next page)

(continued from previous page)

```
                },
                {
                    "name": "server21",
                    "cluster": "cluster2",
                    "port": "9080"
                },
                {
                    "name": "server22",
                    "cluster": "cluster2",
                    "port": "9090"
                }
            ],
            "library": [
                {
                    "name": "lib1",
                    "target": "cluster1"
                },
                {
                    "name": "lib2",
                    "target": "cluster2"
                }
            ]
        }
    }
}
```

To extract all clusters from this structure, you can use the following query:

```
- name: Display all cluster names
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query('domain.cluster[*].name') }}
↪"
```

To extract all server names:

```
- name: Display all server names
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query('domain.server[*].name') }}"
```

To extract ports from cluster1:

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query(server_name_cluster1_query)␣
↪}}"
  vars:
    server_name_cluster1_query: "domain.server[?cluster=='cluster1'].port"
```

Note: You can use a variable to make the query more readable.

To print out the ports from cluster1 in a comma separated string:

```
- name: Display all ports from cluster1 as a string
  ansible.builtin.debug:
    msg: "{{ domain_definition | community.general.json_query('domain.server[?
↪cluster==`cluster1`].port') | join(', ') }}"
```

Note: In the example above, quoting literals using backticks avoids escaping quotes and maintains readability.

You can use YAML single quote escaping:

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query('domain.server[?cluster=='
↪'cluster1''].port') }}"
```

Note: Escaping single quotes within single quotes in YAML is done by doubling the single quote.

To get a hash map with all ports and names of a cluster:

```
- name: Display all server ports and names from cluster1
  ansible.builtin.debug:
    var: item
  loop: "{{ domain_definition | community.general.json_query(server_name_cluster1_query)␣
↪}}"
  vars:
    server_name_cluster1_query: "domain.server[?cluster=='cluster2'].{name: name, port:␣
↪port}"
```

To extract ports from all clusters with name starting with 'server1':

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    msg: "{{ domain_definition | to_json | from_json | community.general.json_
↪query(server_name_query) }}"
  vars:
    server_name_query: "domain.server[?starts_with(name,'server1')].port"
```

To extract ports from all clusters with name containing 'server1':

```
- name: Display all ports from cluster1
  ansible.builtin.debug:
    msg: "{{ domain_definition | to_json | from_json | community.general.json_
↪query(server_name_query) }}"
  vars:
    server_name_query: "domain.server[?contains(name,'server1')].port"
```

Note: while using `starts_with` and `contains`, you have to use `` to_json | from_json `` filter for correct parsing of data structure.

Randomizing data

When you need a randomly generated value, use one of these filters.

Random MAC addresses

New in version 2.6.

This filter can be used to generate a random MAC address from a string prefix.

Note: This filter has migrated to the community.general collection. Follow the installation instructions to install that collection.

To get a random MAC address from a string prefix starting with '52:54:00':

```
"{{ '52:54:00' | community.general.random_mac }}"
# => '52:54:00:ef:1c:03'
```

Note that if anything is wrong with the prefix string, the filter will issue an error.

New in version 2.9.

As of Ansible version 2.9, you can also initialize the random number generator from a seed to create random-but-idempotent MAC addresses:

```
"{{ '52:54:00' | community.general.random_mac(seed=inventory_hostname) }}"
```

Random items or numbers

The `random` filter in Ansible is an extension of the default Jinja2 random filter, and can be used to return a random item from a sequence of items or to generate a random number based on a range.

To get a random item from a list:

```
"{{ ['a','b','c'] | random }}"
# => 'c'
```

To get a random number between 0 (inclusive) and a specified integer (exclusive):

```
"{{ 60 | random }} * * * * root /script/from/cron"
# => '21 * * * * root /script/from/cron'
```

To get a random number from 0 to 100 but in steps of 10:

```
{{ 101 | random(step=10) }}
# => 70
```

To get a random number from 1 to 100 but in steps of 10:

```
{{ 101 | random(1, 10) }}
# => 31
{{ 101 | random(start=1, step=10) }}
# => 51
```

You can initialize the random number generator from a seed to create random-but-idempotent numbers:

```
"{{ 60 | random(seed=inventory_hostname) }} * * * * root /script/from/cron"
```

Shuffling a list

The `shuffle` filter randomizes an existing list, giving a different order every invocation.

To get a random list from an existing list:

```
{{ ['a','b','c'] | shuffle }}
# => ['c','a','b']
{{ ['a','b','c'] | shuffle }}
# => ['b','c','a']
```

You can initialize the shuffle generator from a seed to generate a random-but-idempotent order:

```
{{ ['a','b','c'] | shuffle(seed=inventory_hostname) }}
# => ['b','a','c']
```

The shuffle filter returns a list whenever possible. If you use it with a non 'listable' item, the filter does nothing.

Managing list variables

You can search for the minimum or maximum value in a list, or flatten a multi-level list.

To get the minimum value from list of numbers:

```
{{ list1 | min }}
```

New in version 2.11.

To get the minimum value in a list of objects:

```
{{ [{'val': 1}, {'val': 2}] | min(attribute='val') }}
```

To get the maximum value from a list of numbers:

```
{{ [3, 4, 2] | max }}
```

New in version 2.11.

To get the maximum value in a list of objects:

```
{{ [{'val': 1}, {'val': 2}] | max(attribute='val') }}
```

New in version 2.5.

Flatten a list (same thing the *flatten* lookup does):

```
{{ [3, [4, 2] ] | flatten }}
# => [3, 4, 2]
```

Flatten only the first level of a list (akin to the *items* lookup):

```
{{ [3, [4, [2]] ] | flatten(levels=1) }}
# => [3, 4, [2]]
```

New in version 2.11.

Preserve nulls in a list, by default flatten removes them. :

```
{{ [3, None, [4, [2]] ] | flatten(levels=1, skip_nulls=False) }}
# => [3, None, 4, [2]]
```

Selecting from sets or lists (set theory)

You can select or combine items from sets or lists.

New in version 1.4.

To get a unique set from a list:

```
# list1: [1, 2, 5, 1, 3, 4, 10]
{{ list1 | unique }}
# => [1, 2, 5, 3, 4, 10]
```

To get a union of two lists:

```
# list1: [1, 2, 5, 1, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | union(list2) }}
# => [1, 2, 5, 1, 3, 4, 10, 11, 99]
```

To get the intersection of 2 lists (unique list of all items in both):

```
# list1: [1, 2, 5, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | intersect(list2) }}
# => [1, 2, 5, 3, 4]
```

To get the difference of 2 lists (items in 1 that don't exist in 2):

```
# list1: [1, 2, 5, 1, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | difference(list2) }}
# => [10]
```

To get the symmetric difference of 2 lists (items exclusive to each list):

```
# list1: [1, 2, 5, 1, 3, 4, 10]
# list2: [1, 2, 3, 4, 5, 11, 99]
{{ list1 | symmetric_difference(list2) }}
# => [10, 11, 99]
```

Calculating numbers (math)

New in version 1.9.

You can calculate logs, powers, and roots of numbers with Ansible filters. Jinja2 provides other mathematical functions like abs() and round().

Get the logarithm (default is e):

```
{{ 8 | log }}
# => 2.0794415416798357
```

Get the base 10 logarithm:

```
{{ 8 | log(10) }}
# => 0.9030899869919435
```

Give me the power of 2! (or 5):

```
{{ 8 | pow(5) }}
# => 32768.0
```

Square root, or the 5th:

```
{{ 8 | root }}
# => 2.8284271247461903

{{ 8 | root(5) }}
# => 1.5157165665103982
```

Managing network interactions

These filters help you with common network tasks.

Note: These filters have migrated to the ansible.netcommon collection. Follow the installation instructions to install that collection.

IP address filters

New in version 1.9.

To test if a string is a valid IP address:

```
{{ myvar | ansible.netcommon.ipaddr }}
```

You can also require a specific IP protocol version:

```
{{ myvar | ansible.netcommon.ipv4 }}
{{ myvar | ansible.netcommon.ipv6 }}
```

IP address filter can also be used to extract specific information from an IP address. For example, to get the IP address itself from a CIDR, you can use:

```
{{ '192.0.2.1/24' | ansible.netcommon.ipaddr('address') }}
# => 192.0.2.1
```

More information about `ipaddr` filter and complete usage guide can be found in playbooks_filters_ipaddr.

Network CLI filters

New in version 2.4.

To convert the output of a network device CLI command into structured JSON output, use the `parse_cli` filter:

```
{{ output | ansible.netcommon.parse_cli('path/to/spec') }}
```

The `parse_cli` filter will load the spec file and pass the command output through it, returning JSON output. The YAML spec file defines how to parse the CLI output.

The spec file should be valid formatted YAML. It defines how to parse the CLI output and return JSON data. Below is an example of a valid spec file that will parse the output from the `show vlan` command.

```
---
vars:
  vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }}"
    enabled: "{{ item.state != 'act/lshut' }}"
    state: "{{ item.state }}"

keys:
  vlans:
    value: "{{ vlan }}"
    items: "^(?P<vlan_id>\\d+)\\s+(?P<name>\\w+)\\s+(?P<state>active|act/lshut|suspended)
↪"
  state_static:
    value: present
```

The spec file above will return a JSON data structure that is a list of hashes with the parsed VLAN information.

The same command could be parsed into a hash by using the key and values directives. Here is an example of how to parse the output into a hash value using the same `show vlan` command.

```
---
vars:
  vlan:
    key: "{{ item.vlan_id }}"
    values:
      vlan_id: "{{ item.vlan_id }}"
      name: "{{ item.name }}"
      enabled: "{{ item.state != 'act/lshut' }}"
      state: "{{ item.state }}"

keys:
  vlans:
    value: "{{ vlan }}"
    items: "^(?P<vlan_id>\\d+)\\s+(?P<name>\\w+)\\s+(?P<state>active|act/lshut|suspended)
```

(continues on next page)

(continued from previous page)

```
↪"
  state_static:
    value: present
```

Another common use case for parsing CLI commands is to break a large command into blocks that can be parsed. This can be done using the `start_block` and `end_block` directives to break the command into blocks that can be parsed.

```
---
vars:
  interface:
    name: "{{ item[0].match[0] }}"
    state: "{{ item[1].state }}"
    mode: "{{ item[2].match[0] }}"

keys:
  interfaces:
    value: "{{ interface }}"
    start_block: "^Ethernet.*$"
    end_block: "^$"
    items:
      - "^(?P<name>Ethernet\\d\\/\\d*)"
      - "admin state is (?P<state>.+),"
      - "Port mode is (.+)"
```

The example above will parse the output of `show interface` into a list of hashes.

The network filters also support parsing the output of a CLI command using the TextFSM library. To parse the CLI output with TextFSM use the following filter:

```
{{ output.stdout[0] | ansible.netcommon.parse_cli_textfsm('path/to/fsm') }}
```

Use of the TextFSM filter requires the TextFSM library to be installed.

Network XML filters

New in version 2.5.

To convert the XML output of a network device command into structured JSON output, use the `parse_xml` filter:

```
{{ output | ansible.netcommon.parse_xml('path/to/spec') }}
```

The `parse_xml` filter will load the spec file and pass the command output through formatted as JSON.

The spec file should be valid formatted YAML. It defines how to parse the XML output and return JSON data.

Below is an example of a valid spec file that will parse the output from the `show vlan | display xml` command.

```
---
vars:
  vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }}"
    desc: "{{ item.desc }}"
    enabled: "{{ item.state.get('inactive') != 'inactive' }}"
```

(continues on next page)

(continued from previous page)

```
    state: "{% if item.state.get('inactive') == 'inactive'%} inactive {% else %} active {
↪% endif %}"

keys:
  vlans:
    value: "{{ vlan }}"
    top: configuration/vlans/vlan
    items:
      vlan_id: vlan-id
      name: name
      desc: description
      state: ".[@inactive='inactive']"
```

The spec file above will return a JSON data structure that is a list of hashes with the parsed VLAN information.

The same command could be parsed into a hash by using the key and values directives. Here is an example of how to parse the output into a hash value using the same `show vlan | display xml` command.

```
---
vars:
  vlan:
    key: "{{ item.vlan_id }}"
    values:
        vlan_id: "{{ item.vlan_id }}"
        name: "{{ item.name }}"
        desc: "{{ item.desc }}"
        enabled: "{{ item.state.get('inactive') != 'inactive' }}"
        state: "{% if item.state.get('inactive') == 'inactive'%} inactive {% else %}␣
↪active {% endif %}"

keys:
  vlans:
    value: "{{ vlan }}"
    top: configuration/vlans/vlan
    items:
      vlan_id: vlan-id
      name: name
      desc: description
      state: ".[@inactive='inactive']"
```

The value of `top` is the XPath relative to the XML root node. In the example XML output given below, the value of `top` is `configuration/vlans/vlan`, which is an XPath expression relative to the root node (<rpc-reply>). `configuration` in the value of `top` is the outer most container node, and `vlan` is the inner-most container node.

`items` is a dictionary of key-value pairs that map user-defined names to XPath expressions that select elements. The Xpath expression is relative to the value of the XPath value contained in `top`. For example, the `vlan_id` in the spec file is a user defined name and its value `vlan-id` is the relative to the value of XPath in `top`

Attributes of XML tags can be extracted using XPath expressions. The value of `state` in the spec is an XPath expression used to get the attributes of the `vlan` tag in output XML.:

```
<rpc-reply>
  <configuration>
    <vlans>
```

(continues on next page)

(continued from previous page)

```
      <vlan inactive="inactive">
       <name>vlan-1</name>
       <vlan-id>200</vlan-id>
       <description>This is vlan-1</description>
      </vlan>
    </vlans>
  </configuration>
</rpc-reply>
```

Note: For more information on supported XPath expressions, see XPath Support.

Network VLAN filters

New in version 2.8.

Use the `vlan_parser` filter to transform an unsorted list of VLAN integers into a sorted string list of integers according to IOS-like VLAN list rules. This list has the following properties:

- Vlans are listed in ascending order.
- Three or more consecutive VLANs are listed with a dash.
- The first line of the list can be first_line_len characters long.
- Subsequent list lines can be other_line_len characters.

To sort a VLAN list:

```
{{ [3003, 3004, 3005, 100, 1688, 3002, 3999] | ansible.netcommon.vlan_parser }}
```

This example renders the following sorted list:

```
['100,1688,3002-3005,3999']
```

Another example Jinja template:

```
{% set parsed_vlans = vlans | ansible.netcommon.vlan_parser %}
switchport trunk allowed vlan {{ parsed_vlans[0] }}
{% for i in range (1, parsed_vlans | count) %}
switchport trunk allowed vlan add {{ parsed_vlans[i] }}
{% endfor %}
```

This allows for dynamic generation of VLAN lists on a Cisco IOS tagged interface. You can store an exhaustive raw list of the exact VLANs required for an interface and then compare that to the parsed IOS output that would actually be generated for the configuration.

Hashing and encrypting strings and passwords

New in version 1.9.

To get the sha1 hash of a string:

```
{{ 'test1' | hash('sha1') }}
# => "b444ac06613fc8d63795be9ad0beaf55011936ac"
```

To get the md5 hash of a string:

```
{{ 'test1' | hash('md5') }}
# => "5a105e8b9d40e1329780d62ea2265d8a"
```

Get a string checksum:

```
{{ 'test2' | checksum }}
# => "109f4b3c50d7b0df729d299bc6f8e9ef9066971f"
```

Other hashes (platform dependent):

```
{{ 'test2' | hash('blowfish') }}
```

To get a sha512 password hash (random salt):

```
{{ 'passwordsaresecret' | password_hash('sha512') }}
# => "$6$UIv3676O/ilZzWEE
↪$ktEfFF19NQPF2zyxqxGkAceTnbEgpEKuGBtk6MlU4v2ZorWaVQUMyurgmHCh2Fr4wpmQ/Y.AlXMJkRnIS4RfH/
↪"
```

To get a sha256 password hash with a specific salt:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt') }}
# => "$5$mysecretsalt$ReKNyDYjkKNqRVwouShhsEqZ3VOE8eoVO4exihOfvG4"
```

An idempotent method to generate unique hashes per system is to use a salt that is consistent between runs:

```
{{ 'secretpassword' | password_hash('sha512', 65534 | random(seed=inventory_hostname) |␣
↪string) }}
# => "$6$43927$lQxPKz2M2X.NWO.gK.
↪t7phLwOKQMcSq72XxDZQ0XzYV6DlL1OD72h417aj16OnHTGxNzhftXJQBcjbunLEepM0"
```

Hash types available depend on the control system running Ansible, 'hash' depends on hashlib, password_hash depends on passlib. The crypt is used as a fallback if `passlib` is not installed.

New in version 2.7.

Some hash types allow providing a rounds parameter:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt', rounds=10000) }}
# => "$5$rounds=10000$mysecretsalt$Tkm80llAxD4YHll6AgNIztKn0vzAACsuuEfYeGP7tm7"
```

The filter *password_hash* produces different results depending on whether you installed *passlib* or not.

To ensure idempotency, specify *rounds* to be neither *crypt*'s nor *passlib*'s default, which is *5000* for *crypt* and a variable value (*535000* for sha256, *656000* for sha512) for *passlib*:

```
{{ 'secretpassword' | password_hash('sha256', 'mysecretsalt', rounds=5001) }}
# => "$5$rounds=5001$mysecretsalt$wXcTWWXbfcR8er5IVf7NuquLvnUA6s8/qdtOhAZ.xN."
```

Hash type 'blowfish' (BCrypt) provides the facility to specify the version of the BCrypt algorithm.

```
{{ 'secretpassword' | password_hash('blowfish', '1234567890123456789012', ident='2b') }}
# => "$2b$12$123456789012345678901uuJ4qFdej6xnWjOQT.FStqfdoY8dYUPC"
```

Note: The parameter is only available for blowfish (BCrypt). Other hash types will simply ignore this parameter. Valid values for this parameter are: ['2', '2a', '2y', '2b']

New in version 2.12.

You can also use the Ansible *vault* filter to encrypt data:

```
# simply encrypt my key in a vault
vars:
  myvaultedkey: "{{ keyrawdata|vault(passphrase) }}"

- name: save templated vaulted data
  template: src=dump_template_data.j2 dest=/some/key/vault.txt
  vars:
    mysalt: '{{ 2**256|random(seed=inventory_hostname) }}'
    template_data: '{{ secretdata|vault(vaultsecret, salt=mysalt) }}'
```

And then decrypt it using the unvault filter:

```
# simply decrypt my key from a vault
vars:
  mykey: "{{ myvaultedkey|unvault(passphrase) }}"

- name: save templated unvaulted data
  template: src=dump_template_data.j2 dest=/some/key/clear.txt
  vars:
    template_data: '{{ secretdata|unvault(vaultsecret) }}'
```

Manipulating text

Several filters work with text, including URLs, file names, and path names.

Adding comments to files

The `comment` filter lets you create comments in a file from text in a template, with a variety of comment styles. By default Ansible uses # to start a comment line and adds a blank comment line above and below your comment text. For example the following:

```
{{ "Plain style (default)" | comment }}
```

produces this output:

```
#
# Plain style (default)
#
```

Ansible offers styles for comments in C (`//...`), C block (`/*...*/`), Erlang (`%...`) and XML (`<!--...-->`):

```
{{ "C style" | comment('c') }}
{{ "C block style" | comment('cblock') }}
{{ "Erlang style" | comment('erlang') }}
{{ "XML style" | comment('xml') }}
```

You can define a custom comment character. This filter:

```
{{ "My Special Case" | comment(decoration="! ") }}
```

produces:

```
!
! My Special Case
!
```

You can fully customize the comment style:

```
{{ "Custom style" | comment('plain', prefix='#######\n#', postfix='#\n#######\n   ###\n ␣
↪  #') }}
```

That creates the following output:

```
#######
#
# Custom style
#
#######
   ###
    #
```

The filter can also be applied to any Ansible variable. For example to make the output of the `ansible_managed` variable more readable, we can change the definition in the `ansible.cfg` file to this:

```
[defaults]

ansible_managed = This file is managed by Ansible.%n
  template: {file}
  date: %Y-%m-%d %H:%M:%S
  user: {uid}
  host: {host}
```

and then use the variable with the *comment* filter:

```
{{ ansible_managed | comment }}
```

which produces this output:

```
#
# This file is managed by Ansible.
#
# template: /home/ansible/env/dev/ansible_managed/roles/role1/templates/test.j2
# date: 2015-09-10 11:02:58
# user: ansible
# host: myhost
#
```

URLEncode Variables

The `urlencode` filter quotes data for use in a URL path or query using UTF-8:

```
{{ 'Trollhättan' | urlencode }}
# => 'Trollh%C3%A4ttan'
```

Splitting URLs

New in version 2.4.

The `urlsplit` filter extracts the fragment, hostname, netloc, password, path, port, query, scheme, and username from an URL. With no arguments, returns a dictionary of all the fields:

```
{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('hostname') }}
# => 'www.acme.com'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('netloc') }}
# => 'user:password@www.acme.com:9000'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('username') }}
# => 'user'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('password') }}
# => 'password'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('path') }}
# => '/dir/index.html'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('port') }}
# => '9000'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('scheme') }}
# => 'http'
```

(continues on next page)

(continued from previous page)

```
{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('query') }}
# => 'query=term'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit('fragment') }}
# => 'fragment'

{{ "http://user:password@www.acme.com:9000/dir/index.html?query=term#fragment" |␣
↪urlsplit }}
# =>
#   {
#       "fragment": "fragment",
#       "hostname": "www.acme.com",
#       "netloc": "user:password@www.acme.com:9000",
#       "password": "password",
#       "path": "/dir/index.html",
#       "port": 9000,
#       "query": "query=term",
#       "scheme": "http",
#       "username": "user"
#   }
```

Searching strings with regular expressions

To search in a string or extract parts of a string with a regular expression, use the `regex_search` filter:

```
# Extracts the database name from a string
{{ 'server1/database42' | regex_search('database[0-9]+') }}
# => 'database42'

# Example for a case insensitive search in multiline mode
{{ 'foo\nBAR' | regex_search('^bar', multiline=True, ignorecase=True) }}
# => 'BAR'

# Extracts server and database id from a string
{{ 'server1/database42' | regex_search('server([0-9]+)/database([0-9]+)', '\\1', '\\2') }
↪}
# => ['1', '42']

# Extracts dividend and divisor from a division
{{ '21/42' | regex_search('(?P<dividend>[0-9]+)/(?P<divisor>[0-9]+)', '\\g<dividend>', '\
↪\g<divisor>') }}
# => ['21', '42']
```

The `regex_search` filter returns an empty string if it cannot find a match:

```
{{ 'ansible' | regex_search('foobar') }}
# => ''
```

Note: The `regex_search` filter returns `None` when used in a Jinja expression (for example in conjunction with operators, other filters, and so on). See the two examples below.

```
{{ 'ansible' | regex_search('foobar') == '' }}
# => False
{{ 'ansible' | regex_search('foobar') is none }}
# => True
```

This is due to historic behavior and the custom re-implementation of some of the Jinja internals in Ansible. Enable the `jinja2_native` setting if you want the `regex_search` filter to always return `None` if it cannot find a match. See *Why does the regex_search filter return None instead of an empty string?* for details.

To extract all occurrences of regex matches in a string, use the `regex_findall` filter:

```
# Returns a list of all IPv4 addresses in the string
{{ 'Some DNS servers are 8.8.8.8 and 8.8.4.4' | regex_findall('\\b(?:[0-9]{1,3}\\.){3}[0-
↪9]{1,3}\\b') }}
# => ['8.8.8.8', '8.8.4.4']

# Returns all lines that end with "ar"
{{ 'CAR\ntar\nfoo\nbar\n' | regex_findall('^.ar$', multiline=True, ignorecase=True) }}
# => ['CAR', 'tar', 'bar']
```

To replace text in a string with regex, use the `regex_replace` filter:

```
# Convert "ansible" to "able"
{{ 'ansible' | regex_replace('^a.*i(.*)$', 'a\\1') }}
# => 'able'

# Convert "foobar" to "bar"
{{ 'foobar' | regex_replace('^f.*o(.*)$', '\\1') }}
# => 'bar'

# Convert "localhost:80" to "localhost, 80" using named groups
{{ 'localhost:80' | regex_replace('^(?P<host>.+):(?P<port>\\d+)$', '\\g<host>, \\g<port>
↪') }}
# => 'localhost, 80'

# Convert "localhost:80" to "localhost"
{{ 'localhost:80' | regex_replace(':80') }}
# => 'localhost'

# Comment all lines that end with "ar"
{{ 'CAR\ntar\nfoo\nbar\n' | regex_replace('^(.ar)$', '#\\1', multiline=True,␣
↪ignorecase=True) }}
# => '#CAR\n#tar\nfoo\n#bar\n'
```

Note: If you want to match the whole string and you are using `*` make sure to always wraparound your regular expression with the start/end anchors. For example `^(.*)$` will always match only one result, while `(.*)` on some Python versions will match the whole string and an empty string at the end, which means it will make two replacements:

```
# add "https://" prefix to each item in a list
GOOD:
```

(continues on next page)

(continued from previous page)

```
{{ hosts | map('regex_replace', '^(.*)$', 'https://\\1') | list }}
{{ hosts | map('regex_replace', '(.+)', 'https://\\1') | list }}
{{ hosts | map('regex_replace', '^', 'https://') | list }}

BAD:
{{ hosts | map('regex_replace', '(.*)', 'https://\\1') | list }}

# append ':80' to each item in a list
GOOD:
{{ hosts | map('regex_replace', '^(.*)$', '\\1:80') | list }}
{{ hosts | map('regex_replace', '(.+)', '\\1:80') | list }}
{{ hosts | map('regex_replace', '$', ':80') | list }}

BAD:
{{ hosts | map('regex_replace', '(.*)', '\\1:80') | list }}
```

Note: Prior to ansible 2.0, if `regex_replace` filter was used with variables inside YAML arguments (as opposed to simpler 'key=value' arguments), then you needed to escape backreferences (for example, \\1) with 4 backslashes (\\\\) instead of 2 (\\).

New in version 2.0.

To escape special characters within a standard Python regex, use the `regex_escape` filter (using the default `re_type='python'` option):

```
# convert '^f.*o(.*)$' to '\^f\.\*o\(\.\*\)\$'
{{ '^f.*o(.*)$' | regex_escape() }}
```

New in version 2.8.

To escape special characters within a POSIX basic regex, use the `regex_escape` filter with the `re_type='posix_basic'` option:

```
# convert '^f.*o(.*)$' to '\^f\.\*o(\.\*)\$'
{{ '^f.*o(.*)$' | regex_escape('posix_basic') }}
```

Managing file names and path names

To get the last name of a file path, like 'foo.txt' out of '/etc/asdf/foo.txt':

```
{{ path | basename }}
```

To get the last name of a windows style file path (new in version 2.0):

```
{{ path | win_basename }}
```

To separate the windows drive letter from the rest of a file path (new in version 2.0):

```
{{ path | win_splitdrive }}
```

To get only the windows drive letter:

```
{{ path | win_splitdrive | first }}
```

To get the rest of the path without the drive letter:

```
{{ path | win_splitdrive | last }}
```

To get the directory from a path:

```
{{ path | dirname }}
```

To get the directory from a windows path (new version 2.0):

```
{{ path | win_dirname }}
```

To expand a path containing a tilde (~) character (new in version 1.5):

```
{{ path | expanduser }}
```

To expand a path containing environment variables:

```
{{ path | expandvars }}
```

Note: *expandvars* expands local variables; using it on remote paths can lead to errors.

New in version 2.6.

To get the real path of a link (new in version 1.8):

```
{{ path | realpath }}
```

To get the relative path of a link, from a start point (new in version 1.7):

```
{{ path | relpath('/etc') }}
```

To get the root and extension of a path or file name (new in version 2.0):

```
# with path == 'nginx.conf' the return would be ('nginx', '.conf')
{{ path | splitext }}
```

The `splitext` filter always returns a pair of strings. The individual components can be accessed by using the `first` and `last` filters:

```
# with path == 'nginx.conf' the return would be 'nginx'
{{ path | splitext | first }}

# with path == 'nginx.conf' the return would be '.conf'
{{ path | splitext | last }}
```

To join one or more path components:

```
{{ ('/etc', path, 'subdir', file) | path_join }}
```

New in version 2.10.

Manipulating strings

To add quotes for shell usage:

```
- name: Run a shell command
  ansible.builtin.shell: echo {{ string_value | quote }}
```

To concatenate a list into a string:

```
{{ list | join(" ") }}
```

To split a string into a list:

```
{{ csv_string | split(",") }}
```

New in version 2.11.

To work with Base64 encoded strings:

```
{{ encoded | b64decode }}
{{ decoded | string | b64encode }}
```

As of version 2.6, you can define the type of encoding to use, the default is `utf-8`:

```
{{ encoded | b64decode(encoding='utf-16-le') }}
{{ decoded | string | b64encode(encoding='utf-16-le') }}
```

Note: The `string` filter is only required for Python 2 and ensures that text to encode is a unicode string. Without that filter before b64encode the wrong value will be encoded.

Note: The return value of b64decode is a string. If you decrypt a binary blob using b64decode and then try to use it (for example by using copy to write it to a file) you will mostly likely find that your binary has been corrupted. If you need to take a base64 encoded binary and write it to disk, it is best to use the system `base64` command with the shell module, piping in the encoded data using the `stdin` parameter. For example: `shell:  cmd="base64 --decode > myfile.bin" stdin="{{ encoded }}"`

New in version 2.6.

Managing UUIDs

To create a namespaced UUIDv5:

```
{{ string | to_uuid(namespace='11111111-2222-3333-4444-555555555555') }}
```

New in version 2.10.

To create a namespaced UUIDv5 using the default Ansible namespace '361E6D51-FAEC-444A-9079-341386DA8E2E':

```
{{ string | to_uuid }}
```

New in version 1.9.

To make use of one attribute from each item in a list of complex variables, use the Jinja2 map filter:

```
# get a comma-separated list of the mount points (for example, "/,/mnt/stuff") on a host
{{ ansible_mounts | map(attribute='mount') | join(',') }}
```

Handling dates and times

To get a date object from a string use the *to_datetime* filter:

```
# Get total amount of seconds between two dates. Default date format is %Y-%m-%d %H:%M:
↪%S but you can pass your own format
{{ (("2016-08-14 20:00:12" | to_datetime) - ("2015-12-25" | to_datetime('%Y-%m-%d'))).
↪total_seconds()  }}

# Get remaining seconds after delta has been calculated. NOTE: This does NOT convert␣
↪years, days, hours, and so on to seconds. For that, use total_seconds()
{{ (("2016-08-14 20:00:12" | to_datetime) - ("2016-08-14 18:00:00" | to_datetime)).
↪seconds  }}
# This expression evaluates to "12" and not "132". Delta is 2 hours, 12 seconds

# get amount of days between two dates. This returns only number of days and discards␣
↪remaining hours, minutes, and seconds
{{ (("2016-08-14 20:00:12" | to_datetime) - ("2015-12-25" | to_datetime('%Y-%m-%d'))).
↪days  }}
```

Note: For a full list of format codes for working with python date format strings, see the python datetime documentation.

New in version 2.4.

To format a date using a string (like with the shell date command), use the "strftime" filter:

```
# Display year-month-day
{{ '%Y-%m-%d' | strftime }}
# => "2021-03-19"

# Display hour:min:sec
{{ '%H:%M:%S' | strftime }}
# => "21:51:04"

# Use ansible_date_time.epoch fact
{{ '%Y-%m-%d %H:%M:%S' | strftime(ansible_date_time.epoch) }}
# => "2021-03-19 21:54:09"

# Use arbitrary epoch value
{{ '%Y-%m-%d' | strftime(0) }}          # => 1970-01-01
{{ '%Y-%m-%d' | strftime(1441357287) }} # => 2015-09-04
```

New in version 2.13.

strftime takes an optional utc argument, defaulting to False, meaning times are in the local timezone:

```
{{ '%H:%M:%S' | strftime }}           # time now in local timezone
{{ '%H:%M:%S' | strftime(utc=True) }} # time now in UTC
```

Note: To get all string possibilities, check https://docs.python.org/3/library/time.html#time.strftime

Getting Kubernetes resource names

Note: These filters have migrated to the kubernetes.core collection. Follow the installation instructions to install that collection.

Use the "k8s_config_resource_name" filter to obtain the name of a Kubernetes ConfigMap or Secret, including its hash:

```
{{ configmap_resource_definition | kubernetes.core.k8s_config_resource_name }}
```

This can then be used to reference hashes in Pod specifications:

```
my_secret:
  kind: Secret
  metadata:
    name: my_secret_name

deployment_resource:
  kind: Deployment
  spec:
    template:
      spec:
        containers:
        - envFrom:
            - secretRef:
                name: {{ my_secret | kubernetes.core.k8s_config_resource_name }}
```

New in version 2.8.

See also:

Ansible playbooks
An introduction to playbooks

Conditionals
Conditional statements in playbooks

Using Variables
All about variables

Loops
Looping in playbooks

Roles
Playbook organization by roles

General tips
Tips and tricks for playbooks

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Tests

Tests in Jinja are a way of evaluating template expressions and returning True or False. Jinja ships with many of these. See builtin tests in the official Jinja template documentation.

The main difference between tests and filters are that Jinja tests are used for comparisons, whereas filters are used for data manipulation, and have different applications in jinja. Tests can also be used in list processing filters, like `map()` and `select()` to choose items in the list.

Like all templating, tests always execute on the Ansible controller, **not** on the target of a task, as they test local data.

In addition to those Jinja2 tests, Ansible supplies a few more and users can easily create their own.

Test syntax

Test syntax varies from filter syntax (`variable | filter`). Historically Ansible has registered tests as both jinja tests and jinja filters, allowing for them to be referenced using filter syntax.

As of Ansible 2.5, using a jinja test as a filter will generate a deprecation warning. As of Ansible 2.9+ using jinja test syntax is required.

The syntax for using a jinja test is as follows

```
variable is test_name
```

Such as

```
result is failed
```

Testing strings

To match strings against a substring or a regular expression, use the `match`, `search` or `regex` tests

```
vars:
  url: "https://example.com/users/foo/resources/bar"

tasks:
    - debug:
        msg: "matched pattern 1"
      when: url is match("https://example.com/users/.*/resources")

    - debug:
        msg: "matched pattern 2"
      when: url is search("users/.*/resources/.*")

    - debug:
        msg: "matched pattern 3"
      when: url is search("users")

    - debug:
        msg: "matched pattern 4"
      when: url is regex("example\.com/\w+/foo")
```

`match` succeeds if it finds the pattern at the beginning of the string, while `search` succeeds if it finds the pattern anywhere within string. By default, `regex` works like `search`, but `regex` can be configured to perform other tests as well, by passing the `match_type` keyword argument. In particular, `match_type` determines the `re` method that gets used to perform the search. The full list can be found in the relevant Python documentation here.

All of the string tests also take optional `ignorecase` and `multiline` arguments. These correspond to `re.I` and `re.M` from Python's `re` library, respectively.

Vault

New in version 2.10.

You can test whether a variable is an inline single vault encrypted value using the `vault_encrypted` test.

```
vars:
  variable: !vault |
    $ANSIBLE_VAULT;1.2;AES256;dev
    61323931353866666336306139373937316366366138656131323863373866376666353364373761
    3539633234313836346435323766306164626134376564330a373530313635343535343133316133
    36643666306434616266376434363239346433643238336464643566386135356334303736353136
    6565633133366366360a326566323363363936613664616364623437336130623133343530333739
    3039

tasks:
  - debug:
```

(continues on next page)

(continued from previous page)

```
    msg: '{{ (variable is vault_encrypted) | ternary("Vault encrypted", "Not vault␣
↪encrypted") }}'
```

Testing truthiness

New in version 2.10.

As of Ansible 2.10, you can now perform Python like truthy and falsy checks.

```
- debug:
    msg: "Truthy"
  when: value is truthy
  vars:
    value: "some string"

- debug:
    msg: "Falsy"
  when: value is falsy
  vars:
    value: ""
```

Additionally, the `truthy` and `falsy` tests accept an optional parameter called `convert_bool` that will attempt to convert boolean indicators to actual booleans.

```
- debug:
    msg: "Truthy"
  when: value is truthy(convert_bool=True)
  vars:
    value: "yes"

- debug:
    msg: "Falsy"
  when: value is falsy(convert_bool=True)
  vars:
    value: "off"
```

Comparing versions

New in version 1.6.

Note: In 2.5 `version_compare` was renamed to `version`

To compare a version number, such as checking if the `ansible_facts['distribution_version']` version is greater than or equal to '12.04', you can use the `version` test.

The `version` test can also be used to evaluate the `ansible_facts['distribution_version']`

```
{{ ansible_facts['distribution_version'] is version('12.04', '>=') }}
```

If `ansible_facts['distribution_version']` is greater than or equal to 12.04, this test returns True, otherwise False.

The `version` test accepts the following operators

```
<, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne
```

This test also accepts a 3rd parameter, `strict` which defines if strict version parsing as defined by `ansible.module_utils.compat.version.StrictVersion` should be used. The default is `False` (using `ansible.module_utils.compat.version.LooseVersion`), `True` enables strict version parsing

```
{{ sample_version_var is version('1.0', operator='lt', strict=True) }}
```

As of Ansible 2.11 the `version` test accepts a `version_type` parameter which is mutually exclusive with `strict`, and accepts the following values

```
loose, strict, semver, semantic, pep440
```

loose
: This type corresponds to the Python `distutils.version.LooseVersion` class. All version formats are valid for this type. The rules for comparison are simple and predictable, but may not always give expected results.

strict
: This type corresponds to the Python `distutils.version.StrictVersion` class. A version number consists of two or three dot-separated numeric components, with an optional "pre-release" tag on the end. The pre-release tag consists of a single letter 'a' or 'b' followed by a number. If the numeric components of two version numbers are equal, then one with a pre-release tag will always be deemed earlier (lesser) than one without.

semver/semantic
: This type implements the Semantic Version scheme for version comparison.

pep440
: This type implements the Python PEP-440 versioning rules for version comparison. Added in version 2.14.

Using `version_type` to compare a semantic version would be achieved like the following

```
{{ sample_semver_var is version('2.0.0-rc.1+build.123', 'lt', version_type='semver') }}
```

In Ansible 2.14, the `pep440` option for `version_type` was added, and the rules of this type are defined in PEP-440. The following example showcases how this type can differentiate pre-releases as being less than a general release.

```
{{ '2.14.0rc1' is version('2.14.0', 'lt', version_type='pep440') }}
```

When using `version` in a playbook or role, don't use `{{ }}` as described in the FAQ

```
vars:
    my_version: 1.2.3

tasks:
    - debug:
        msg: "my_version is higher than 1.0.0"
      when: my_version is version('1.0.0', '>')
```

Set theory tests

New in version 2.1.

Note: In 2.5 `issubset` and `issuperset` were renamed to `subset` and `superset`

To see if a list includes or is included by another list, you can use ‘subset’ and ‘superset’

```
vars:
    a: [1,2,3,4,5]
    b: [2,3]
tasks:
    - debug:
        msg: "A includes B"
      when: a is superset(b)

    - debug:
        msg: "B is included in A"
      when: b is subset(a)
```

Testing if a list contains a value

New in version 2.8.

Ansible includes a `contains` test which operates similarly, but in reverse of the Jinja2 provided `in` test. The `contains` test is designed to work with the `select`, `reject`, `selectattr`, and `rejectattr` filters

```
vars:
  lacp_groups:
    - master: lacp0
      network: 10.65.100.0/24
      gateway: 10.65.100.1
      dns4:
        - 10.65.100.10
        - 10.65.100.11
      interfaces:
        - em1
        - em2

    - master: lacp1
      network: 10.65.120.0/24
      gateway: 10.65.120.1
      dns4:
        - 10.65.100.10
        - 10.65.100.11
      interfaces:
          - em3
          - em4

tasks:
  - debug:
      msg: "{{ (lacp_groups|selectattr('interfaces', 'contains', 'em1')|first).master }}"
```

Testing if a list value is True

New in version 2.4.

You can use *any* and *all* to check if any or all elements in a list are true or not

```
vars:
  mylist:
      - 1
      - "{{ 3 == 3 }}"
      - True
  myotherlist:
      - False
      - True
tasks:

  - debug:
      msg: "all are true!"
    when: mylist is all

  - debug:
      msg: "at least one is true"
    when: myotherlist is any
```

Testing paths

Note: In 2.5 the following tests were renamed to remove the `is_` prefix

The following tests can provide information about a path on the controller

```
- debug:
    msg: "path is a directory"
  when: mypath is directory

- debug:
    msg: "path is a file"
  when: mypath is file

- debug:
    msg: "path is a symlink"
  when: mypath is link

- debug:
    msg: "path already exists"
  when: mypath is exists

- debug:
    msg: "path is {{ (mypath is abs)|ternary('absolute','relative')}}"

- debug:
    msg: "path is the same file as path2"
```

(continues on next page)

(continued from previous page)

```
  when: mypath is same_file(path2)

- debug:
    msg: "path is a mount"
  when: mypath is mount

- debug:
    msg: "path is a directory"
  when: mypath is directory
  vars:
     mypath: /my/patth

- debug:
    msg: "path is a file"
  when: "'/my/path' is file"
```

Testing size formats

The `human_readable` and `human_to_bytes` functions let you test your playbooks to make sure you are using the right size format in your tasks, and that you provide Byte format to computers and human-readable format to people.

Human readable

Asserts whether the given string is human readable or not.

For example

```
- name: "Human Readable"
  assert:
    that:
      - '"1.00 Bytes" == 1|human_readable'
      - '"1.00 bits" == 1|human_readable(isbits=True)'
      - '"10.00 KB" == 10240|human_readable'
      - '"97.66 MB" == 102400000|human_readable'
      - '"0.10 GB" == 102400000|human_readable(unit="G")'
      - '"0.10 Gb" == 102400000|human_readable(isbits=True, unit="G")'
```

This would result in

```
{ "changed": false, "msg": "All assertions passed" }
```

Human to bytes

Returns the given string in the Bytes format.

For example

```
- name: "Human to Bytes"
  assert:
    that:
      - "{{'0'|human_to_bytes}}        == 0"
      - "{{'0.1'|human_to_bytes}}      == 0"
      - "{{'0.9'|human_to_bytes}}      == 1"
      - "{{'1'|human_to_bytes}}        == 1"
      - "{{'10.00 KB'|human_to_bytes}} == 10240"
      - "{{   '11 MB'|human_to_bytes}} == 11534336"
      - "{{  '1.1 GB'|human_to_bytes}} == 1181116006"
      - "{{'10.00 Kb'|human_to_bytes(isbits=True)}} == 10240"
```

This would result in

```
{ "changed": false, "msg": "All assertions passed" }
```

Testing task results

The following tasks are illustrative of the tests meant to check the status of tasks

```
tasks:

  - shell: /usr/bin/foo
    register: result
    ignore_errors: True

  - debug:
      msg: "it failed"
    when: result is failed

  # in most cases you'll want a handler, but if you want to do something right now, this␣
↪is nice
  - debug:
      msg: "it changed"
    when: result is changed

  - debug:
      msg: "it succeeded in Ansible >= 2.1"
    when: result is succeeded

  - debug:
      msg: "it succeeded"
    when: result is success

  - debug:
      msg: "it was skipped"
    when: result is skipped
```

Note: From 2.1, you can also use success, failure, change, and skip so that the grammar matches, for those who need to be strict about it.

Type Tests

When looking to determine types, it may be tempting to use the `type_debug` filter and compare that to the string name of that type, however, you should instead use type test comparisons, such as:

```
tasks:
  - name: "String interpretation"
    vars:
      a_string: "A string"
      a_dictionary: {"a": "dictionary"}
      a_list: ["a", "list"]
    assert:
      that:
      # Note that a string is classed as also being "iterable" and "sequence", but not
↪"mapping"
      - a_string is string and a_string is iterable and a_string is sequence and a_
↪string is not mapping

      # Note that a dictionary is classed as not being a "string", but is "iterable",
↪"sequence" and "mapping"
      - a_dictionary is not string and a_dictionary is iterable and a_dictionary is␣
↪mapping

      # Note that a list is classed as not being a "string" or "mapping" but is "iterable
↪" and "sequence"
      - a_list is not string and a_list is not mapping and a_list is iterable

  - name: "Number interpretation"
    vars:
      a_float: 1.01
      a_float_as_string: "1.01"
      an_integer: 1
      an_integer_as_string: "1"
    assert:
      that:
      # Both a_float and an_integer are "number", but each has their own type as well
      - a_float is number and a_float is float
      - an_integer is number and an_integer is integer

      # Both a_float_as_string and an_integer_as_string are not numbers
      - a_float_as_string is not number and a_float_as_string is string
      - an_integer_as_string is not number and a_float_as_string is string

      # a_float or a_float_as_string when cast to a float and then to a string should␣
↪match the same value cast only to a string
      - a_float | float | string == a_float | string
      - a_float_as_string | float | string == a_float_as_string | string
```

(continues on next page)

(continued from previous page)

```
      # Likewise an_integer and an_integer_as_string when cast to an integer and then to␣
↪a string should match the same value cast only to an integer
      - an_integer | int | string == an_integer | string
      - an_integer_as_string | int | string == an_integer_as_string | string

      # However, a_float or a_float_as_string cast as an integer and then a string does␣
↪not match the same value cast to a string
      - a_float | int | string != a_float | string
      - a_float_as_string | int | string != a_float_as_string | string

      # Again, Likewise an_integer and an_integer_as_string cast as a float and then a␣
↪string does not match the same value cast to a string
      - an_integer | float | string != an_integer | string
      - an_integer_as_string | float | string != an_integer_as_string | string

  - name: "Native Boolean interpretation"
    loop:
    - yes
    - true
    - True
    - TRUE
    - no
    - No
    - NO
    - false
    - False
    - FALSE
    assert:
      that:
      # Note that while other values may be cast to boolean values, these are the only␣
↪ones which are natively considered boolean
      # Note also that `yes` is the only case sensitive variant of these values.
      - item is boolean
```

See also:

Ansible playbooks
An introduction to playbooks

Conditionals
Conditional statements in playbooks

Using Variables
All about variables

Loops
Looping in playbooks

Roles
Playbook organization by roles

General tips
Tips and tricks for playbooks

User Mailing List

Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Lookups

Lookup plugins retrieve data from outside sources such as files, databases, key/value stores, APIs, and other services. Like all templating, lookups execute and are evaluated on the Ansible control machine. Ansible makes the data returned by a lookup plugin available using the standard templating system. Before Ansible 2.5, lookups were mostly used indirectly in `with_<lookup>` constructs for looping. Starting with Ansible 2.5, lookups are used more explicitly as part of Jinja2 expressions fed into the `loop` keyword.

Using lookups in variables

You can populate variables using lookups. Ansible evaluates the value each time it is executed in a task (or template).

```
vars:
  motd_value: "{{ lookup('file', '/etc/motd') }}"
tasks:
  - debug:
      msg: "motd value is {{ motd_value }}"
```

For more details and a list of lookup plugins in ansible-core, see *Working with plugins*. You may also find lookup plugins in collections. You can review a list of lookup plugins installed on your control machine with the command `ansible-doc -l -t lookup`.

See also:

Working with playbooks
An introduction to playbooks

Conditionals
Conditional statements in playbooks

Using Variables
All about variables

Loops
Looping in playbooks

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Python3 in templates

Ansible uses Jinja2 to take advantage of Python data types and standard functions in templates and variables. You can use these data types and standard functions to perform a rich set of operations on your data. However, if you use templates, you must be aware of differences between Python versions.

These topics help you design templates that work on both Python2 and Python3. They might also help if you are upgrading from Python2 to Python3. Upgrading within Python2 or Python3 does not usually introduce changes that affect Jinja2 templates.

Dictionary views

In Python2, the `dict.keys()`, `dict.values()`, and `dict.items()` methods return a list. Jinja2 returns that to Ansible using a string representation that Ansible can turn back into a list.

In Python3, those methods return a dictionary view object. The string representation that Jinja2 returns for dictionary views cannot be parsed back into a list by Ansible. It is, however, easy to make this portable by using the `list` filter whenever using `dict.keys()`, `dict.values()`, or `dict.items()`.

```
vars:
  hosts:
    testhost1: 127.0.0.2
    testhost2: 127.0.0.3
tasks:
  - debug:
      msg: '{{ item }}'
    # Only works with Python 2
    #loop: "{{ hosts.keys() }}"
    # Works with both Python 2 and Python 3
    loop: "{{ hosts.keys() | list }}"
```

dict.iteritems()

Python2 dictionaries have `iterkeys()`, `itervalues()`, and `iteritems()` methods.

Python3 dictionaries do not have these methods. Use `dict.keys()`, `dict.values()`, and `dict.items()` to make your playbooks and templates compatible with both Python2 and Python3.

```
vars:
  hosts:
    testhost1: 127.0.0.2
    testhost2: 127.0.0.3
tasks:
  - debug:
      msg: '{{ item }}'
    # Only works with Python 2
    #loop: "{{ hosts.iteritems() }}"
    # Works with both Python 2 and Python 3
    loop: "{{ hosts.items() | list }}"
```

See also:

- The *Dictionary views* entry for information on why the `list filter` is necessary here.

The now function: get the current time

New in version 2.8.

The `now()` Jinja2 function retrieves a Python datetime object or a string representation for the current time.

The `now()` function supports 2 arguments:

utc
Specify `True` to get the current time in UTC. Defaults to `False`.

fmt
Accepts a strftime string that returns a formatted date time string.

Loops

Ansible offers the `loop`, `with_<lookup>`, and `until` keywords to execute a task multiple times. Examples of commonly-used loops include changing ownership on several files and/or directories with the file module, creating multiple users with the user module, and repeating a polling step until a certain result is reached.

Note:

- We added `loop` in Ansible 2.5. It is not yet a full replacement for `with_<lookup>`, but we recommend it for most use cases.
- We have not deprecated the use of `with_<lookup>` - that syntax will still be valid for the foreseeable future.
- We are looking to improve `loop` syntax - watch this page and the changelog for updates.

Comparing `loop` and `with_*`

- The `with_<lookup>` keywords rely on *Lookup plugins* - even `items` is a lookup.
- The `loop` keyword is equivalent to `with_list`, and is the best choice for simple loops.
- The `loop` keyword will not accept a string as input, see *Ensuring list input for loop: using query rather than lookup*.
- Generally speaking, any use of `with_*` covered in *Migrating from with_X to loop* can be updated to use `loop`.
- Be careful when changing `with_items` to `loop`, as `with_items` performed implicit single-level flattening. You may need to use `flatten(1)` with `loop` to match the exact outcome. For example, to get the same output as:

```
with_items:
  - 1
  - [2,3]
  - 4
```

you would need

```
loop: "{{ [1, [2, 3], 4] | flatten(1) }}"
```

- Any `with_*` statement that requires using `lookup` within a loop should not be converted to use the `loop` keyword. For example, instead of doing:

```
loop: "{{ lookup('fileglob', '*.txt', wantlist=True) }}"
```

it's cleaner to keep

```
with_fileglob: '*.txt'
```

Standard loops

Iterating over a simple list

Repeated tasks can be written as standard loops over a simple list of strings. You can define the list directly in the task.

```
- name: Add several users
  ansible.builtin.user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
     - testuser1
     - testuser2
```

You can define the list in a variables file, or in the 'vars' section of your play, then refer to the name of the list in the task.

```
loop: "{{ somelist }}"
```

Either of these examples would be the equivalent of

```
- name: Add user testuser1
  ansible.builtin.user:
    name: "testuser1"
    state: present
    groups: "wheel"

- name: Add user testuser2
  ansible.builtin.user:
    name: "testuser2"
    state: present
    groups: "wheel"
```

You can pass a list directly to a parameter for some plugins. Most of the packaging modules, like yum and apt, have this capability. When available, passing the list to a parameter is better than looping over the task. For example

```
- name: Optimal yum
  ansible.builtin.yum:
    name: "{{ list_of_packages }}"
    state: present

- name: Non-optimal yum, slower and may cause issues with interdependencies
  ansible.builtin.yum:
    name: "{{ item }}"
    state: present
  loop: "{{ list_of_packages }}"
```

Check the module documentation to see if you can pass a list to any particular module's parameter(s).

Iterating over a list of hashes

If you have a list of hashes, you can reference subkeys in a loop. For example:

```
- name: Add several users
  ansible.builtin.user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }
```

When combining *conditionals* with a loop, the `when:` statement is processed separately for each item. See *Basic conditionals with when* for examples.

Iterating over a dictionary

To loop over a dict, use the *dict2items*:

```
- name: Using dict2items
  ansible.builtin.debug:
    msg: "{{ item.key }} - {{ item.value }}"
  loop: "{{ tag_data | dict2items }}"
  vars:
    tag_data:
      Environment: dev
      Application: payment
```

Here, we are iterating over *tag_data* and printing the key and the value from it.

Registering variables with a loop

You can register the output of a loop as a variable. For example

```
- name: Register loop output as a variable
  ansible.builtin.shell: "echo {{ item }}"
  loop:
    - "one"
    - "two"
  register: echo
```

When you use `register` with a loop, the data structure placed in the variable will contain a `results` attribute that is a list of all responses from the module. This differs from the data structure returned when using `register` without a loop.

```
{
    "changed": true,
    "msg": "All items completed",
    "results": [
        {
            "changed": true,
```

(continues on next page)

(continued from previous page)

```
            "cmd": "echo \"one\" ",
            "delta": "0:00:00.003110",
            "end": "2013-12-19 12:00:05.187153",
            "invocation": {
                "module_args": "echo \"one\"",
                "module_name": "shell"
            },
            "item": "one",
            "rc": 0,
            "start": "2013-12-19 12:00:05.184043",
            "stderr": "",
            "stdout": "one"
        },
        {
            "changed": true,
            "cmd": "echo \"two\" ",
            "delta": "0:00:00.002920",
            "end": "2013-12-19 12:00:05.245502",
            "invocation": {
                "module_args": "echo \"two\"",
                "module_name": "shell"
            },
            "item": "two",
            "rc": 0,
            "start": "2013-12-19 12:00:05.242582",
            "stderr": "",
            "stdout": "two"
        }
    ]
}
```

Subsequent loops over the registered variable to inspect the results may look like

```
- name: Fail if return code is not 0
  ansible.builtin.fail:
    msg: "The command ({{ item.cmd }}) did not have a 0 return code"
  when: item.rc != 0
  loop: "{{ echo.results }}"
```

During iteration, the result of the current item will be placed in the variable.

```
- name: Place the result of the current item in the variable
  ansible.builtin.shell: echo "{{ item }}"
  loop:
    - one
    - two
  register: echo
  changed_when: echo.stdout != "one"
```

Complex loops

Iterating over nested lists

You can use Jinja2 expressions to iterate over complex lists. For example, a loop can combine nested lists.

```
- name: Give users access to multiple databases
  community.mysql.mysql_user:
    name: "{{ item[0] }}"
    priv: "{{ item[1] }}.*:ALL"
    append_privs: true
    password: "foo"
  loop: "{{ ['alice', 'bob'] | product(['clientdb', 'employeedb', 'providerdb']) | list }
↪}"
```

Retrying a task until a condition is met

New in version 1.4.

You can use the `until` keyword to retry a task until a certain condition is met. Here's an example:

```
- name: Retry a task until a certain condition is met
  ansible.builtin.shell: /usr/bin/foo
  register: result
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

This task runs up to 5 times with a delay of 10 seconds between each attempt. If the result of any attempt has "all systems go" in its stdout, the task succeeds. The default value for "retries" is 3 and "delay" is 5.

To see the results of individual retries, run the play with `-vv`.

When you run a task with `until` and register the result as a variable, the registered variable will include a key called "attempts", which records the number of the retries for the task.

Note: You must set the `until` parameter if you want a task to retry. If `until` is not defined, the value for the `retries` parameter is forced to 1.

Looping over inventory

To loop over your inventory, or just a subset of it, you can use a regular `loop` with the `ansible_play_batch` or `groups` variables.

```
- name: Show all the hosts in the inventory
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ groups['all'] }}"

- name: Show all the hosts in the current play
  ansible.builtin.debug:
```

(continues on next page)

(continued from previous page)

```
    msg: "{{ item }}"
  loop: "{{ ansible_play_batch }}"
```

There is also a specific lookup plugin `inventory_hostnames` that can be used like this

```
- name: Show all the hosts in the inventory
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ query('inventory_hostnames', 'all') }}"

- name: Show all the hosts matching the pattern, ie all but the group www
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ query('inventory_hostnames', 'all:!www') }}"
```

More information on the patterns can be found in *Patterns: targeting hosts and groups*.

Ensuring list input for `loop`: using `query` rather than `lookup`

The `loop` keyword requires a list as input, but the `lookup` keyword returns a string of comma-separated values by default. Ansible 2.5 introduced a new Jinja2 function named *query* that always returns a list, offering a simpler interface and more predictable output from lookup plugins when using the `loop` keyword.

You can force `lookup` to return a list to `loop` by using `wantlist=True`, or you can use `query` instead.

The following two examples do the same thing.

```
loop: "{{ query('inventory_hostnames', 'all') }}"

loop: "{{ lookup('inventory_hostnames', 'all', wantlist=True) }}"
```

Adding controls to loops

New in version 2.1.

The `loop_control` keyword lets you manage your loops in useful ways.

Limiting loop output with `label`

New in version 2.2.

When looping over complex data structures, the console output of your task can be enormous. To limit the displayed output, use the `label` directive with `loop_control`.

```
- name: Create servers
  digital_ocean:
    name: "{{ item.name }}"
    state: present
  loop:
    - name: server1
      disks: 3gb
```

(continues on next page)

(continued from previous page)

```
      ram: 15Gb
      network:
        nic01: 100Gb
        nic02: 10Gb
        ...
  loop_control:
    label: "{{ item.name }}"
```

The output of this task will display just the `name` field for each `item` instead of the entire contents of the multi-line `{{ item }}` variable.

Note: This is for making console output more readable, not protecting sensitive data. If there is sensitive data in `loop`, set `no_log: yes` on the task to prevent disclosure.

Pausing within a loop

New in version 2.2.

To control the time (in seconds) between the execution of each item in a task loop, use the `pause` directive with `loop_control`.

```
# main.yml
- name: Create servers, pause 3s before creating next
  community.digitalocean.digital_ocean:
    name: "{{ item }}"
    state: present
  loop:
    - server1
    - server2
  loop_control:
    pause: 3
```

Tracking progress through a loop with `index_var`

New in version 2.5.

To keep track of where you are in a loop, use the `index_var` directive with `loop_control`. This directive specifies a variable name to contain the current loop index.

```
- name: Count our fruit
  ansible.builtin.debug:
    msg: "{{ item }} with index {{ my_idx }}"
  loop:
    - apple
    - banana
    - pear
  loop_control:
    index_var: my_idx
```

Note: *index_var* is 0 indexed.

Defining inner and outer variable names with `loop_var`

New in version 2.1.

You can nest two looping tasks using `include_tasks`. However, by default Ansible sets the loop variable `item` for each loop. This means the inner, nested loop will overwrite the value of `item` from the outer loop. You can specify the name of the variable for each loop using `loop_var` with `loop_control`.

```yaml
# main.yml
- include_tasks: inner.yml
  loop:
    - 1
    - 2
    - 3
  loop_control:
    loop_var: outer_item

# inner.yml
- name: Print outer and inner items
  ansible.builtin.debug:
    msg: "outer item={{ outer_item }} inner item={{ item }}"
  loop:
    - a
    - b
    - c
```

Note: If Ansible detects that the current loop is using a variable which has already been defined, it will raise an error to fail the task.

Extended loop variables

New in version 2.8.

As of Ansible 2.8 you can get extended loop information using the `extended` option to loop control. This option will expose the following information.

Variable	Description
`ansible_loop.allitems`	The list of all items in the loop
`ansible_loop.index`	The current iteration of the loop. (1 indexed)
`ansible_loop.index0`	The current iteration of the loop. (0 indexed)
`ansible_loop.revindex`	The number of iterations from the end of the loop (1 indexed)
`ansible_loop.revindex0`	The number of iterations from the end of the loop (0 indexed)
`ansible_loop.first`	`True` if first iteration
`ansible_loop.last`	`True` if last iteration
`ansible_loop.length`	The number of items in the loop
`ansible_loop.previtem`	The item from the previous iteration of the loop. Undefined during the first iteration.
`ansible_loop.nextitem`	The item from the following iteration of the loop. Undefined during the last iteration.

```
loop_control:
  extended: true
```

Note: When using `loop_control.extended` more memory will be utilized on the control node. This is a result of `ansible_loop.allitems` containing a reference to the full loop data for every loop. When serializing the results for display in callback plugins within the main ansible process, these references may be dereferenced causing memory usage to increase.

New in version 2.14.

To disable the `ansible_loop.allitems` item, to reduce memory consumption, set `loop_control.extended_allitems: no`.

```
loop_control:
  extended: true
  extended_allitems: false
```

Accessing the name of your loop_var

New in version 2.8.

As of Ansible 2.8 you can get the name of the value provided to `loop_control.loop_var` using the `ansible_loop_var` variable

For role authors, writing roles that allow loops, instead of dictating the required `loop_var` value, you can gather the value through the following

```
"{{ lookup('vars', ansible_loop_var) }}"
```

Migrating from with_X to loop

In most cases, loops work best with the `loop` keyword instead of `with_X` style loops. The `loop` syntax is usually best expressed using filters instead of more complex use of `query` or `lookup`.

These examples show how to convert many common `with_` style loops to `loop` and filters.

with_list

`with_list` is directly replaced by `loop`.

```
- name: with_list
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_list:
    - one
    - two

- name: with_list -> loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop:
    - one
    - two
```

with_items

`with_items` is replaced by `loop` and the `flatten` filter.

```
- name: with_items
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_items: "{{ items }}"

- name: with_items -> loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
```

with_indexed_items

`with_indexed_items` is replaced by `loop`, the `flatten` filter and `loop_control.index_var`.

```
- name: with_indexed_items
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_indexed_items: "{{ items }}"

- name: with_indexed_items -> loop
  ansible.builtin.debug:
```

(continues on next page)

(continued from previous page)

```
    msg: "{{ index }} - {{ item }}"
  loop: "{{ items|flatten(levels=1) }}"
  loop_control:
    index_var: index
```

with_flattened

`with_flattened` is replaced by `loop` and the `flatten` filter.

```
- name: with_flattened
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_flattened: "{{ items }}"

- name: with_flattened -> loop
  ansible.builtin.debug:
    msg: "{{ item }}"
  loop: "{{ items|flatten }}"
```

with_together

`with_together` is replaced by `loop` and the `zip` filter.

```
- name: with_together
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_together:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_together -> loop
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ list_one|zip(list_two)|list }}"
```

Another example with complex data

```
- name: with_together -> loop
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }} - {{ item.2 }}"
  loop: "{{ data[0]|zip(*data[1:])|list }}"
  vars:
    data:
      - ['a', 'b', 'c']
      - ['d', 'e', 'f']
      - ['g', 'h', 'i']
```

with_dict

`with_dict` can be substituted by `loop` and either the `dictsort` or `dict2items` filters.

```
- name: with_dict
  ansible.builtin.debug:
    msg: "{{ item.key }} - {{ item.value }}"
  with_dict: "{{ dictionary }}"

- name: with_dict -> loop (option 1)
  ansible.builtin.debug:
    msg: "{{ item.key }} - {{ item.value }}"
  loop: "{{ dictionary|dict2items }}"

- name: with_dict -> loop (option 2)
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ dictionary|dictsort }}"
```

with_sequence

`with_sequence` is replaced by `loop` and the `range` function, and potentially the `format` filter.

```
- name: with_sequence
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_sequence: start=0 end=4 stride=2 format=testuser%02x

- name: with_sequence -> loop
  ansible.builtin.debug:
    msg: "{{ 'testuser%02x' | format(item) }}"
  loop: "{{ range(0, 4 + 1, 2)|list }}"
```

The range of the loop is exclusive of the end point.

with_subelements

`with_subelements` is replaced by `loop` and the `subelements` filter.

```
- name: with_subelements
  ansible.builtin.debug:
    msg: "{{ item.0.name }} - {{ item.1 }}"
  with_subelements:
    - "{{ users }}"
    - mysql.hosts

- name: with_subelements -> loop
  ansible.builtin.debug:
    msg: "{{ item.0.name }} - {{ item.1 }}"
  loop: "{{ users|subelements('mysql.hosts') }}"
```

with_nested/with_cartesian

`with_nested` and `with_cartesian` are replaced by loop and the `product` filter.

```
- name: with_nested
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  with_nested:
    - "{{ list_one }}"
    - "{{ list_two }}"

- name: with_nested -> loop
  ansible.builtin.debug:
    msg: "{{ item.0 }} - {{ item.1 }}"
  loop: "{{ list_one|product(list_two)|list }}"
```

with_random_choice

`with_random_choice` is replaced by just use of the `random` filter, without need of `loop`.

```
- name: with_random_choice
  ansible.builtin.debug:
    msg: "{{ item }}"
  with_random_choice: "{{ my_list }}"

- name: with_random_choice -> loop (No loop is needed here)
  ansible.builtin.debug:
    msg: "{{ my_list|random }}"
  tags: random
```

See also:

Ansible playbooks
    An introduction to playbooks

Roles
    Playbook organization by roles

General tips
    Tips and tricks for playbooks

Conditionals
    Conditional statements in playbooks

Using Variables
    All about variables

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Controlling where tasks run: delegation and local actions

By default Ansible gathers facts and executes all tasks on the machines that match the `hosts` line of your playbook. This page shows you how to delegate tasks to a different machine or group, delegate facts to specific machines or groups, or run an entire playbook locally. Using these approaches, you can manage inter-related environments precisely and efficiently. For example, when updating your webservers, you might need to remove them from a load-balanced pool temporarily. You cannot perform this task on the webservers themselves. By delegating the task to localhost, you keep all the tasks within the same play.

Tasks that cannot be delegated

Some tasks always execute on the controller. These tasks, including `include`, `add_host`, and `debug`, cannot be delegated.

Delegating tasks

If you want to perform a task on one host with reference to other hosts, use the `delegate_to` keyword on a task. This is ideal for managing nodes in a load balanced pool or for controlling outage windows. You can use delegation with the *serial* keyword to control the number of hosts executing at one time:

```yaml
---
- hosts: webservers
  serial: 5

  tasks:
    - name: Take out of load balancer pool
      ansible.builtin.command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

    - name: Actual steps would go here
      ansible.builtin.yum:
        name: acme-web-stack
        state: latest

    - name: Add back to load balancer pool
      ansible.builtin.command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1
```

The first and third tasks in this play run on 127.0.0.1, which is the machine running Ansible. There is also a shorthand syntax that you can use on a per-task basis: `local_action`. Here is the same playbook as above, but using the shorthand syntax for delegating to 127.0.0.1:

```
---
# ...

  tasks:
    - name: Take out of load balancer pool
      local_action: ansible.builtin.command /usr/bin/take_out_of_pool {{ inventory_
↪hostname }}

# ...

    - name: Add back to load balancer pool
      local_action: ansible.builtin.command /usr/bin/add_back_to_pool {{ inventory_
↪hostname }}
```

You can use a local action to call 'rsync' to recursively copy files to the managed servers:

```
---
# ...

  tasks:
    - name: Recursively copy files from management server to target
      local_action: ansible.builtin.command rsync -a /path/to/files {{ inventory_
↪hostname }}:/path/to/target/
```

Note that you must have passphrase-less SSH keys or an ssh-agent configured for this to work, otherwise rsync asks for a passphrase.

To specify more arguments, use the following syntax:

```
---
# ...

  tasks:
    - name: Send summary mail
      local_action:
        module: community.general.mail
        subject: "Summary Mail"
        to: "{{ mail_recipient }}"
        body: "{{ mail_body }}"
      run_once: True
```

Note:

- The *ansible_host* variable and other connection variables, if present, reflects information about the host a task is delegated to, not the inventory_hostname.

Warning: Although you can `delegate_to` a host that does not exist in inventory (by adding IP address, DNS name or whatever requirement the connection plugin has), doing so does not add the host to your inventory and might cause issues. Hosts delegated to in this way do not inherit variables from the "all" group', so variables like connection user and key are missing. If you must `delegate_to` a non-inventory host, use the add host module.

Delegation and parallel execution

By default Ansible tasks are executed in parallel. Delegating a task does not change this and does not handle concurrency issues (multiple forks writing to the same file). Most commonly, users are affected by this when updating a single file on a single delegated to host for all hosts (using the `copy`, `template`, or `lineinfile` modules, for example). They will still operate in parallel forks (default 5) and overwrite each other.

This can be handled in several ways:

```
- name: "handle concurrency with a loop on the hosts with `run_once: true`"
  lineinfile: "<options here>"
  run_once: true
  loop: '{{ ansible_play_hosts_all }}'
```

By using an intermediate play with *serial: 1* or using *throttle: 1* at task level, for more detail see *Controlling playbook execution: strategies and more*

Delegating facts

Delegating Ansible tasks is like delegating tasks in the real world - your groceries belong to you, even if someone else delivers them to your home. Similarly, any facts gathered by a delegated task are assigned by default to the *inventory_hostname* (the current host), not to the host which produced the facts (the delegated to host). To assign gathered facts to the delegated host instead of the current host, set `delegate_facts` to `true`:

```
---
- hosts: app_servers

  tasks:
    - name: Gather facts from db servers
      ansible.builtin.setup:
      delegate_to: "{{ item }}"
      delegate_facts: true
      loop: "{{ groups['dbservers'] }}"
```

This task gathers facts for the machines in the dbservers group and assigns the facts to those machines, even though the play targets the app_servers group. This way you can lookup *hostvars['dbhost1']['ansible_default_ipv4']['address']* even though dbservers were not part of the play, or left out by using *–limit*.

Local playbooks

It may be useful to use a playbook locally on a remote host, rather than by connecting over SSH. This can be useful for assuring the configuration of a system by putting a playbook in a crontab. This may also be used to run a playbook inside an OS installer, such as an Anaconda kickstart.

To run an entire playbook locally, just set the `hosts:` line to `hosts:  127.0.0.1` and then run the playbook like so:

```
ansible-playbook playbook.yml --connection=local
```

Alternatively, a local connection can be used in a single playbook play, even if other plays in the playbook use the default remote connection type:

```
---
- hosts: 127.0.0.1
  connection: local
```

Note: If you set the connection to local and there is no ansible_python_interpreter set, modules will run under /usr/bin/python and not under {{ ansible_playbook_python }}. Be sure to set ansible_python_interpreter: "{{ ansible_playbook_python }}" in host_vars/localhost.yml, for example. You can avoid this issue by using `local_action` or `delegate_to: localhost` instead.

See also:

Ansible playbooks
    An introduction to playbooks

Controlling playbook execution: strategies and more
    More ways to control how and where Ansible executes

Ansible Examples on GitHub
    Many examples of full-stack deployments

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Conditionals

In a playbook, you may want to execute different tasks, or have different goals, depending on the value of a fact (data about the remote system), a variable, or the result of a previous task. You may want the value of some variables to depend on the value of other variables. Or you may want to create additional groups of hosts based on whether the hosts match other criteria. You can do all of these things with conditionals.

Ansible uses Jinja2 *tests* and *filters* in conditionals. Ansible supports all the standard tests and filters, and adds some unique ones as well.

Note: There are many options to control execution flow in Ansible. You can find more examples of supported conditionals at https://jinja.palletsprojects.com/en/latest/templates/#comparisons.

Basic conditionals with `when`

The simplest conditional statement applies to a single task. Create the task, then add a `when` statement that applies a test. The `when` clause is a raw Jinja2 expression without double curly braces (see group_by_module). When you run the task or playbook, Ansible evaluates the test for all hosts. On any host where the test passes (returns a value of True), Ansible runs that task. For example, if you are installing mysql on multiple machines, some of which have SELinux enabled, you might have a task to configure SELinux to allow mysql to run. You would only want that task to run on machines that have SELinux enabled:

```
tasks:
  - name: Configure SELinux to start mysql on any port
    ansible.posix.seboolean:
      name: mysql_connect_any
      state: true
      persistent: true
    when: ansible_selinux.status == "enabled"
    # all variables can be used directly in conditionals without double curly braces
```

Conditionals based on ansible_facts

Often you want to execute or skip a task based on facts. Facts are attributes of individual hosts, including IP address, operating system, the status of a filesystem, and many more. With conditionals based on facts:

- You can install a certain package only when the operating system is a particular version.
- You can skip configuring a firewall on hosts with internal IP addresses.
- You can perform cleanup tasks only when a filesystem is getting full.

See *Commonly-used facts* for a list of facts that frequently appear in conditional statements. Not all facts exist for all hosts. For example, the 'lsb_major_release' fact used in an example below only exists when the lsb_release package is installed on the target host. To see what facts are available on your systems, add a debug task to your playbook:

```
- name: Show facts available on the system
  ansible.builtin.debug:
    var: ansible_facts
```

Here is a sample conditional based on a fact:

```
tasks:
  - name: Shut down Debian flavored systems
    ansible.builtin.command: /sbin/shutdown -t now
    when: ansible_facts['os_family'] == "Debian"
```

If you have multiple conditions, you can group them with parentheses:

```
tasks:
  - name: Shut down CentOS 6 and Debian 7 systems
    ansible.builtin.command: /sbin/shutdown -t now
    when: (ansible_facts['distribution'] == "CentOS" and ansible_facts['distribution_
↪major_version'] == "6") or
          (ansible_facts['distribution'] == "Debian" and ansible_facts['distribution_
↪major_version'] == "7")
```

You can use logical operators to combine conditions. When you have multiple conditions that all need to be true (that is, a logical and), you can specify them as a list:

```
tasks:
  - name: Shut down CentOS 6 systems
    ansible.builtin.command: /sbin/shutdown -t now
    when:
      - ansible_facts['distribution'] == "CentOS"
      - ansible_facts['distribution_major_version'] == "6"
```

If a fact or variable is a string, and you need to run a mathematical comparison on it, use a filter to ensure that Ansible reads the value as an integer:

```
tasks:
  - ansible.builtin.shell: echo "only on Red Hat 6, derivatives, and later"
    when: ansible_facts['os_family'] == "RedHat" and ansible_facts['lsb']['major_release
↪'] | int >= 6
```

Conditions based on registered variables

Often in a playbook you want to execute or skip a task based on the outcome of an earlier task. For example, you might want to configure a service after it is upgraded by an earlier task. To create a conditional based on a registered variable:

1. Register the outcome of the earlier task as a variable.
2. Create a conditional test based on the registered variable.

You create the name of the registered variable using the `register` keyword. A registered variable always contains the status of the task that created it as well as any output that task generated. You can use registered variables in templates and action lines as well as in conditional when statements. You can access the string contents of the registered variable using `variable.stdout`. For example:

```
- name: Test play
  hosts: all

  tasks:
```

(continues on next page)

(continued from previous page)

```
    - name: Register a variable
      ansible.builtin.shell: cat /etc/motd
      register: motd_contents

    - name: Use the variable in conditional statement
      ansible.builtin.shell: echo "motd contains the word hi"
      when: motd_contents.stdout.find('hi') != -1
```

You can use registered results in the loop of a task if the variable is a list. If the variable is not a list, you can convert it into a list, with either `stdout_lines` or with `variable.stdout.split()`. You can also split the lines by other fields:

```
- name: Registered variable usage as a loop list
  hosts: all
  tasks:

    - name: Retrieve the list of home directories
      ansible.builtin.command: ls /home
      register: home_dirs

    - name: Add home dirs to the backup spooler
      ansible.builtin.file:
        path: /mnt/bkspool/{{ item }}
        src: /home/{{ item }}
        state: link
      loop: "{{ home_dirs.stdout_lines }}"
      # same as loop: "{{ home_dirs.stdout.split() }}"
```

The string content of a registered variable can be empty. If you want to run another task only on hosts where the stdout of your registered variable is empty, check the registered variable's string contents for emptiness:

```
- name: check registered variable for emptiness
  hosts: all

  tasks:

      - name: List contents of directory
        ansible.builtin.command: ls mydir
        register: contents

      - name: Check contents for emptiness
        ansible.builtin.debug:
          msg: "Directory is empty"
        when: contents.stdout == ""
```

Ansible always registers something in a registered variable for every host, even on hosts where a task fails or Ansible skips a task because a condition is not met. To run a follow-up task on these hosts, query the registered variable for `is skipped` (not for "undefined" or "default"). See *Registering variables* for more information. Here are sample conditionals based on the success or failure of a task. Remember to ignore errors if you want Ansible to continue executing on a host when a failure occurs:

```
tasks:
  - name: Register a variable, ignore errors and continue
```

(continues on next page)

(continued from previous page)

```
    ansible.builtin.command: /bin/false
    register: result
    ignore_errors: true

  - name: Run only if the task that registered the "result" variable fails
    ansible.builtin.command: /bin/something
    when: result is failed

  - name: Run only if the task that registered the "result" variable succeeds
    ansible.builtin.command: /bin/something_else
    when: result is succeeded

  - name: Run only if the task that registered the "result" variable is skipped
    ansible.builtin.command: /bin/still/something_else
    when: result is skipped

  - name: Run only if the task that registered the "result" variable changed something.
    ansible.builtin.command: /bin/still/something_else
    when: result is changed
```

Note: Older versions of Ansible used `success` and `fail`, but `succeeded` and `failed` use the correct tense. All of these options are now valid.

Conditionals based on variables

You can also create conditionals based on variables defined in the playbooks or inventory. Because conditionals require boolean input (a test must evaluate as True to trigger the condition), you must apply the `| bool` filter to non boolean variables, such as string variables with content like 'yes', 'on', '1', or 'true'. You can define variables like this:

```
vars:
  epic: true
  monumental: "yes"
```

With the variables above, Ansible would run one of these tasks and skip the other:

```
tasks:
    - name: Run the command if "epic" or "monumental" is true
      ansible.builtin.shell: echo "This certainly is epic!"
      when: epic or monumental | bool

    - name: Run the command if "epic" is false
      ansible.builtin.shell: echo "This certainly isn't epic!"
      when: not epic
```

If a required variable has not been set, you can skip or fail using Jinja2's *defined* test. For example:

```
tasks:
    - name: Run the command if "foo" is defined
      ansible.builtin.shell: echo "I've got '{{ foo }}' and am not afraid to use it!"
      when: foo is defined
```

(continues on next page)

(continued from previous page)

```
    - name: Fail if "bar" is undefined
      ansible.builtin.fail: msg="Bailing out. This play requires 'bar'"
      when: bar is undefined
```

This is especially useful in combination with the conditional import of vars files (see below). As the examples show, you do not need to use *{{ }}* to use variables inside conditionals, as these are already implied.

Using conditionals in loops

If you combine a `when` statement with a *loop*, Ansible processes the condition separately for each item. This is by design, so you can execute the task on some items in the loop and skip it on other items. For example:

```
tasks:
    - name: Run with items greater than 5
      ansible.builtin.command: echo {{ item }}
      loop: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

If you need to skip the whole task when the loop variable is undefined, use the *|default* filter to provide an empty iterator. For example, when looping over a list:

```
- name: Skip the whole task when a loop variable is undefined
  ansible.builtin.command: echo {{ item }}
  loop: "{{ mylist|default([]) }}"
  when: item > 5
```

You can do the same thing when looping over a dict:

```
- name: The same as above using a dict
  ansible.builtin.command: echo {{ item.key }}
  loop: "{{ query('dict', mydict|default({})) }}"
  when: item.value > 5
```

Loading custom facts

You can provide your own facts, as described in *Should you develop a module?*. To run them, just make a call to your own custom fact gathering module at the top of your list of tasks, and variables returned there will be accessible to future tasks:

```
tasks:
    - name: Gather site specific fact data
      action: site_facts

    - name: Use a custom fact
      ansible.builtin.command: /usr/bin/thingy
      when: my_custom_fact_just_retrieved_from_the_remote_system == '1234'
```

Conditionals with re-use

You can use conditionals with re-usable tasks files, playbooks, or roles. Ansible executes these conditional statements differently for dynamic re-use (includes) and for static re-use (imports). See *Re-using Ansible artifacts* for more information on re-use in Ansible.

Conditionals with imports

When you add a conditional to an import statement, Ansible applies the condition to all tasks within the imported file. This behavior is the equivalent of *Tag inheritance: adding tags to multiple tasks*. Ansible applies the condition to every task, and evaluates each task separately. For example, if you want to define and then display a variable that was not previously defined, you might have a playbook called `main.yml` and a tasks file called `other_tasks.yml`:

```
# all tasks within an imported file inherit the condition from the import statement
# main.yml
- hosts: all
  tasks:
  - import_tasks: other_tasks.yml # note "import"
    when: x is not defined

# other_tasks.yml
- name: Set a variable
  ansible.builtin.set_fact:
    x: foo

- name: Print a variable
  ansible.builtin.debug:
    var: x
```

Ansible expands this at execution time to the equivalent of:

```
- name: Set a variable if not defined
  ansible.builtin.set_fact:
    x: foo
  when: x is not defined
  # this task sets a value for x

- name: Do the task if "x" is not defined
  ansible.builtin.debug:
    var: x
  when: x is not defined
  # Ansible skips this task, because x is now defined
```

If `x` is initially defined, both tasks are skipped as intended. But if `x` is initially undefined, the debug task will be skipped since the conditional is evaluated for every imported task. The conditional will evaluate to `true` for the `set_fact` task, which will define the variable and cause the `debug` conditional to evaluate to `false`.

If this is not the behavior you want, use an `include_*` statement to apply a condition only to that statement itself.

```
# using a conditional on include_* only applies to the include task itself
# main.yml
- hosts: all
  tasks:
```

(continues on next page)

(continued from previous page)

```
  - include_tasks: other_tasks.yml # note "include"
    when: x is not defined
```

Now if `x` is initially undefined, the debug task will not be skipped because the conditional is evaluated at the time of the include and does not apply to the individual tasks.

You can apply conditions to `import_playbook` as well as to the other `import_*` statements. When you use this approach, Ansible returns a 'skipped' message for every task on every host that does not match the criteria, creating repetitive output. In many cases the group_by module can be a more streamlined way to accomplish the same objective; see *Handling OS and distro differences*.

Conditionals with includes

When you use a conditional on an `include_*` statement, the condition is applied only to the include task itself and not to any other tasks within the included file(s). To contrast with the example used for conditionals on imports above, look at the same playbook and tasks file, but using an include instead of an import:

```
# Includes let you re-use a file to define a variable when it is not already defined

# main.yml
- include_tasks: other_tasks.yml
  when: x is not defined

# other_tasks.yml
- name: Set a variable
  ansible.builtin.set_fact:
    x: foo

- name: Print a variable
  ansible.builtin.debug:
    var: x
```

Ansible expands this at execution time to the equivalent of:

```
# main.yml
- include_tasks: other_tasks.yml
  when: x is not defined
  # if condition is met, Ansible includes other_tasks.yml

# other_tasks.yml
- name: Set a variable
  ansible.builtin.set_fact:
    x: foo
  # no condition applied to this task, Ansible sets the value of x to foo

- name: Print a variable
  ansible.builtin.debug:
    var: x
  # no condition applied to this task, Ansible prints the debug statement
```

By using `include_tasks` instead of `import_tasks`, both tasks from `other_tasks.yml` will be executed as expected. For more information on the differences between `include` v `import` see *Re-using Ansible artifacts*.

Conditionals with roles

There are three ways to apply conditions to roles:

- Add the same condition or conditions to all tasks in the role by placing your `when` statement under the `roles` keyword. See the example in this section.
- Add the same condition or conditions to all tasks in the role by placing your `when` statement on a static `import_role` in your playbook.
- Add a condition or conditions to individual tasks or blocks within the role itself. This is the only approach that allows you to select or skip some tasks within the role based on your `when` statement. To select or skip tasks within the role, you must have conditions set on individual tasks or blocks, use the dynamic `include_role` in your playbook, and add the condition or conditions to the include. When you use this approach, Ansible applies the condition to the include itself plus any tasks in the role that also have that `when` statement.

When you incorporate a role in your playbook statically with the `roles` keyword, Ansible adds the conditions you define to all the tasks in the role. For example:

```
- hosts: webservers
  roles:
     - role: debian_stock_config
       when: ansible_facts['os_family'] == 'Debian'
```

Selecting variables, files, or templates based on facts

Sometimes the facts about a host determine the values you want to use for certain variables or even the file or template you want to select for that host. For example, the names of packages are different on CentOS and on Debian. The configuration files for common services are also different on different OS flavors and versions. To load different variables file, templates, or other files based on a fact about the hosts:

1) name your vars files, templates, or files to match the Ansible fact that differentiates them
2) select the correct vars file, template, or file for each host with a variable based on that Ansible fact

Ansible separates variables from tasks, keeping your playbooks from turning into arbitrary code with nested conditionals. This approach results in more streamlined and auditable configuration rules because there are fewer decision points to track.

Selecting variables files based on facts

You can create a playbook that works on multiple platforms and OS versions with a minimum of syntax by placing your variable values in vars files and conditionally importing them. If you want to install Apache on some CentOS and some Debian servers, create variables files with YAML keys and values. For example:

```
---
# for vars/RedHat.yml
apache: httpd
somethingelse: 42
```

Then import those variables files based on the facts you gather on the hosts in your playbook:

```
---
- hosts: webservers
```

(continues on next page)

(continued from previous page)

```
  remote_user: root
  vars_files:
    - "vars/common.yml"
    - [ "vars/{{ ansible_facts['os_family'] }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: Make sure apache is started
    ansible.builtin.service:
      name: '{{ apache }}'
      state: started
```

Ansible gathers facts on the hosts in the webservers group, then interpolates the variable "ansible_facts['os_family']" into a list of filenames. If you have hosts with Red Hat operating systems (CentOS, for example), Ansible looks for 'vars/RedHat.yml'. If that file does not exist, Ansible attempts to load 'vars/os_defaults.yml'. For Debian hosts, Ansible first looks for 'vars/Debian.yml', before falling back on 'vars/os_defaults.yml'. If no files in the list are found, Ansible raises an error.

Selecting files and templates based on facts

You can use the same approach when different OS flavors or versions require different configuration files or templates. Select the appropriate file or template based on the variables assigned to each host. This approach is often much cleaner than putting a lot of conditionals into a single template to cover multiple OS or package versions.

For example, you can template out a configuration file that is very different between, say, CentOS and Debian:

```
- name: Template a file
  ansible.builtin.template:
    src: "{{ item }}"
    dest: /etc/myapp/foo.conf
  loop: "{{ query('first_found', { 'files': myfiles, 'paths': mypaths}) }}"
  vars:
    myfiles:
      - "{{ ansible_facts['distribution'] }}.conf"
      -  default.conf
    mypaths: ['search_location_one/somedir/', '/opt/other_location/somedir/']
```

Debugging conditionals

If your conditional `when` statement is not behaving as you intended, you can add a `debug` statement to determine if the condition evaluates to `true` or `false`. A common cause of unexpected behavior in conditionals is testing an integer as a string or a string as an integer. To debug a conditional statement, add the entire statement as the `var:` value in a `debug` task. Ansible then shows the test and how the statement evaluates. For example, here is a set of tasks and sample output:

```
- name: check value of return code
  ansible.builtin.debug:
    var: bar_status.rc

- name: check test for rc value as string
  ansible.builtin.debug:
    var: bar_status.rc == "127"
```

(continues on next page)

(continued from previous page)

```
- name: check test for rc value as integer
  ansible.builtin.debug:
    var: bar_status.rc == 127
```

```
TASK [check value of return code]␣
↪*********************************************************************************
ok: [foo-1] => {
    "bar_status.rc": "127"
}

TASK [check test for rc value as string]␣
↪**************************************************************************
ok: [foo-1] => {
    "bar_status.rc == \"127\"": false
}

TASK [check test for rc value as integer]␣
↪*************************************************************************
ok: [foo-1] => {
    "bar_status.rc == 127": true
}
```

Commonly-used facts

The following Ansible facts are frequently used in conditionals.

ansible_facts['distribution']

Possible values (sample, not complete list):

```
Alpine
Altlinux
Amazon
Archlinux
ClearLinux
Coreos
CentOS
Debian
Fedora
Gentoo
Mandriva
NA
OpenWrt
OracleLinux
RedHat
Slackware
SLES
SMGL
```

(continues on next page)

(continued from previous page)

```
SUSE
Ubuntu
VMwareESX
```

ansible_facts['distribution_major_version']

The major version of the operating system. For example, the value is *16* for Ubuntu 16.04.

ansible_facts['os_family']

Possible values (sample, not complete list):

```
AIX
Alpine
Altlinux
Archlinux
Darwin
Debian
FreeBSD
Gentoo
HP-UX
Mandrake
RedHat
SGML
Slackware
Solaris
Suse
Windows
```

See also:

Working with playbooks
An introduction to playbooks

Roles
Playbook organization by roles

General tips
Tips and tricks for playbooks

Using Variables
All about variables

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Blocks

Blocks create logical groups of tasks. Blocks also offer ways to handle task errors, similar to exception handling in many programming languages.

Grouping tasks with blocks

All tasks in a block inherit directives applied at the block level. Most of what you can apply to a single task (with the exception of loops) can be applied at the block level, so blocks make it much easier to set data or directives common to the tasks. The directive does not affect the block itself, it is only inherited by the tasks enclosed by a block. For example, a *when* statement is applied to the tasks within a block, not to the block itself.

Listing 1: Block example with named tasks inside the block

```
tasks:
  - name: Install, configure, and start Apache
    block:
      - name: Install httpd and memcached
        ansible.builtin.yum:
          name:
          - httpd
          - memcached
          state: present

      - name: Apply the foo config template
        ansible.builtin.template:
          src: templates/src.j2
          dest: /etc/foo.conf

      - name: Start service bar and enable it
        ansible.builtin.service:
          name: bar
          state: started
          enabled: True
    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
    ignore_errors: true
```

In the example above, the 'when' condition will be evaluated before Ansible runs each of the three tasks in the block. All three tasks also inherit the privilege escalation directives, running as the root user. Finally, `ignore_errors: true` ensures that Ansible continues to execute the playbook even if some of the tasks fail.

Names for blocks have been available since Ansible 2.3. We recommend using names in all tasks, within blocks or elsewhere, for better visibility into the tasks being executed when you run the playbook.

Handling errors with blocks

You can control how Ansible responds to task errors using blocks with `rescue` and `always` sections.

Rescue blocks specify tasks to run when an earlier task in a block fails. This approach is similar to exception handling in many programming languages. Ansible only runs rescue blocks after a task returns a 'failed' state. Bad task definitions and unreachable hosts will not trigger the rescue block.

Listing 2: Block error handling example

```
tasks:
- name: Handle the error
  block:
    - name: Print a message
      ansible.builtin.debug:
        msg: 'I execute normally'

    - name: Force a failure
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - name: Print when errors
      ansible.builtin.debug:
        msg: 'I caught an error, can do stuff here to fix it, :-)'
```

You can also add an `always` section to a block. Tasks in the `always` section run no matter what the task status of the previous block is.

Listing 3: Block with always section

```
- name: Always do X
  block:
    - name: Print a message
      ansible.builtin.debug:
        msg: 'I execute normally'

    - name: Force a failure
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I never execute :-('
  always:
    - name: Always do this
      ansible.builtin.debug:
        msg: "This always executes, :-)"
```

Together, these elements offer complex error handling.

Listing 4: Block with all sections

```yaml
- name: Attempt and graceful roll back demo
  block:
    - name: Print a message
      ansible.builtin.debug:
        msg: 'I execute normally'

    - name: Force a failure
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - name: Print when errors
      ansible.builtin.debug:
        msg: 'I caught an error'

    - name: Force a failure in middle of recovery! >:-)
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I also never execute :-('
  always:
    - name: Always do this
      ansible.builtin.debug:
        msg: "This always executes"
```

The tasks in the `block` execute normally. If any tasks in the block return `failed`, the `rescue` section executes tasks to recover from the error. The `always` section runs regardless of the results of the `block` and `rescue` sections.

If an error occurs in the block and the rescue task succeeds, Ansible reverts the failed status of the original task for the run and continues to run the play as if the original task had succeeded. The rescued task is considered successful, and does not trigger `max_fail_percentage` or `any_errors_fatal` configurations. However, Ansible still reports a failure in the playbook statistics.

You can use blocks with `flush_handlers` in a rescue task to ensure that all handlers run even if an error occurs:

Listing 5: Block run handlers in error handling

```yaml
tasks:
  - name: Attempt and graceful roll back demo
    block:
      - name: Print a message
        ansible.builtin.debug:
          msg: 'I execute normally'
        changed_when: true
        notify: Run me even after an error

      - name: Force a failure
        ansible.builtin.command: /bin/false
    rescue:
```

(continues on next page)

(continued from previous page)

```
      - name: Make sure all handlers run
        meta: flush_handlers
handlers:
  - name: Run me even after an error
    ansible.builtin.debug:
      msg: 'This handler runs even on error'
```

New in version 2.1.

Ansible provides a couple of variables for tasks in the `rescue` portion of a block:

ansible_failed_task
: The task that returned 'failed' and triggered the rescue. For example, to get the name use `ansible_failed_task.name`.

ansible_failed_result
: The captured return result of the failed task that triggered the rescue. This would equate to having used this var in the `register` keyword.

Note: In `ansible-core` 2.14 or later, both variables are propagated from an inner block to an outer `rescue` portion of a block.

See also:

Ansible playbooks
: An introduction to playbooks

Roles
: Playbook organization by roles

User Mailing List
: Have a question? Stop by the google group!

Real-time chat
: How to join Ansible chat channels

Handlers: running operations on change

Sometimes you want a task to run only when a change is made on a machine. For example, you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged. Ansible uses handlers to address this use case. Handlers are tasks that only run when notified.

Handler example

This playbook, `verify-apache.yml`, contains a single play with a handler.

```yaml
---
- name: Verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest

    - name: Write the apache config file
      ansible.builtin.template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
      notify:
      - Restart apache

    - name: Ensure apache is running
      ansible.builtin.service:
        name: httpd
        state: started

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: httpd
        state: restarted
```

In this example playbook, the Apache server is restarted by the handler after all tasks complete in the play.

Notifying handlers

Tasks can instruct one or more handlers to execute using the `notify` keyword. The `notify` keyword can be applied to a task and accepts a list of handler names that are notified on a task change. Alternately, a string containing a single handler name can be supplied as well. The following example demonstrates how multiple handlers can be notified by a single task:

```yaml
tasks:
- name: Template configuration file
  ansible.builtin.template:
    src: template.j2
    dest: /etc/foo.conf
```

(continues on next page)

(continued from previous page)

```
  notify:
    - Restart apache
    - Restart memcached

handlers:
  - name: Restart memcached
    ansible.builtin.service:
      name: memcached
      state: restarted

  - name: Restart apache
    ansible.builtin.service:
      name: apache
      state: restarted
```

In the above example the handlers are executed on task change in the following order: `Restart memcached`, `Restart apache`. Handlers are executed in the order they are defined in the `handlers` section, not in the order listed in the `notify` statement. Notifying the same handler multiple times will result in executing the handler only once regardless of how many tasks notify it. For example, if multiple tasks update a configuration file and notify a handler to restart Apache, Ansible only bounces Apache once to avoid unnecessary restarts.

Naming handlers

Handlers must be named in order for tasks to be able to notify them using the `notify` keyword.

Alternately, handlers can utilize the `listen` keyword. Using this handler keyword, handlers can listen on topics that can group multiple handlers as follows:

```
tasks:
  - name: Restart everything
    command: echo "this task will restart the web services"
    notify: "restart web services"

handlers:
  - name: Restart memcached
    service:
      name: memcached
      state: restarted
    listen: "restart web services"

  - name: Restart apache
    service:
      name: apache
      state: restarted
    listen: "restart web services"
```

Notifying the `restart web services` topic results in executing all handlers listening to that topic regardless of how those handlers are named.

This use makes it much easier to trigger multiple handlers. It also decouples handlers from their names, making it easier to share handlers among playbooks and roles (especially when using third-party roles from a shared source such as Ansible Galaxy).

Each handler should have a globally unique name. If multiple handlers are defined with the same name, only the last one defined is notified with `notify`, effectively shadowing all of the previous handlers with the same name. Alternately handlers sharing the same name can all be notified and executed if they listen on the same topic by notifying that topic.

There is only one global scope for handlers (handler names and listen topics) regardless of where the handlers are defined. This also includes handlers defined in roles.

Controlling when handlers run

By default, handlers run after all the tasks in a particular play have been completed. Notified handlers are executed automatically after each of the following sections, in the following order: `pre_tasks`, `roles`/`tasks` and `post_tasks`. This approach is efficient, because the handler only runs once, regardless of how many tasks notify it. For example, if multiple tasks update a configuration file and notify a handler to restart Apache, Ansible only bounces Apache once to avoid unnecessary restarts.

If you need handlers to run before the end of the play, add a task to flush them using the meta module, which executes Ansible actions:

```yaml
tasks:
  - name: Some tasks go here
    ansible.builtin.shell: ...

  - name: Flush handlers
    meta: flush_handlers

  - name: Some other tasks
    ansible.builtin.shell: ...
```

The `meta: flush_handlers` task triggers any handlers that have been notified at that point in the play.

Once handlers are executed, either automatically after each mentioned section or manually by the `flush_handlers` meta task, they can be notified and run again in later sections of the play.

Using variables with handlers

You may want your Ansible handlers to use variables. For example, if the name of a service varies slightly by distribution, you want your output to show the exact name of the restarted service for each target machine. Avoid placing variables in the name of the handler. Since handler names are templated early on, Ansible may not have a value available for a handler name like this:

```yaml
handlers:
# This handler name may cause your play to fail!
- name: Restart "{{ web_service_name }}"
```

If the variable used in the handler name is not available, the entire play fails. Changing that variable mid-play **will not** result in newly created handler.

Instead, place variables in the task parameters of your handler. You can load the values using `include_vars` like this:

```yaml
tasks:
  - name: Set host variables based on distribution
    include_vars: "{{ ansible_facts.distribution }}.yml"

handlers:
```

(continues on next page)

(continued from previous page)

```
- name: Restart web service
  ansible.builtin.service:
    name: "{{ web_service_name | default('httpd') }}"
    state: restarted
```

While handler names can contain a template, `listen` topics cannot.

Handlers in roles

Handlers from roles are not just contained in their roles but rather inserted into global scope with all other handlers from a play. As such they can be used outside of the role they are defined in. It also means that their name can conflict with handlers from outside the role. To ensure that a handler from a role is notified as opposed to one from outside the role with the same name, notify the handler by using its name in the following form: `role_name : handler_name`.

Handlers notified within the `roles` section are automatically flushed at the end of the `tasks` section, but before any `tasks` handlers.

Includes and imports in handlers

Notifying a dynamic include such as `include_task` as a handler results in executing all tasks from within the include. It is not possible to notify a handler defined inside a dynamic include.

Having a static include such as `import_task` as a handler results in that handler being effectively rewritten by handlers from within that import before the play execution. A static include itself cannot be notified; the tasks from within that include, on the other hand, can be notified individually.

Meta tasks as handlers

Since Ansible 2.14 meta tasks are allowed to be used and notified as handlers. Note that however `flush_handlers` cannot be used as a handler to prevent unexpected behavior.

Limitations

A handler cannot run `import_role` or `include_role`.

Error handling in playbooks

When Ansible receives a non-zero return code from a command or a failure from a module, by default it stops executing on that host and continues on other hosts. However, in some circumstances you may want different behavior. Sometimes a non-zero return code indicates success. Sometimes you want a failure on one host to stop execution on all hosts. Ansible provides tools and settings to handle these situations and help you get the behavior, output, and reporting you want.

Ignoring failed commands

By default Ansible stops executing tasks on a host when a task fails on that host. You can use `ignore_errors` to continue on in spite of the failure.

```
- name: Do not count this as a failure
  ansible.builtin.command: /bin/false
  ignore_errors: true
```

The `ignore_errors` directive only works when the task is able to run and returns a value of 'failed'. It does not make Ansible ignore undefined variable errors, connection failures, execution issues (for example, missing packages), or syntax errors.

Ignoring unreachable host errors

New in version 2.7.

You can ignore a task failure due to the host instance being 'UNREACHABLE' with the `ignore_unreachable` keyword. Ansible ignores the task errors, but continues to execute future tasks against the unreachable host. For example, at the task level:

```
- name: This executes, fails, and the failure is ignored
  ansible.builtin.command: /bin/true
  ignore_unreachable: true

- name: This executes, fails, and ends the play for this host
  ansible.builtin.command: /bin/true
```

And at the playbook level:

```
- hosts: all
  ignore_unreachable: true
  tasks:
  - name: This executes, fails, and the failure is ignored
    ansible.builtin.command: /bin/true

  - name: This executes, fails, and ends the play for this host
    ansible.builtin.command: /bin/true
    ignore_unreachable: false
```

Resetting unreachable hosts

If Ansible cannot connect to a host, it marks that host as 'UNREACHABLE' and removes it from the list of active hosts for the run. You can use *meta: clear_host_errors* to reactivate all hosts, so subsequent tasks can try to reach them again.

Handlers and failure

Ansible runs *handlers* at the end of each play. If a task notifies a handler but another task fails later in the play, by default the handler does *not* run on that host, which may leave the host in an unexpected state. For example, a task could update a configuration file and notify a handler to restart some service. If a task later in the same play fails, the configuration file might be changed but the service will not be restarted.

You can change this behavior with the `--force-handlers` command-line option, by including `force_handlers: True` in a play, or by adding `force_handlers = True` to ansible.cfg. When handlers are forced, Ansible will run all notified handlers on all hosts, even hosts with failed tasks. (Note that certain errors could still prevent the handler from running, such as a host becoming unreachable.)

Defining failure

Ansible lets you define what "failure" means in each task using the `failed_when` conditional. As with all conditionals in Ansible, lists of multiple `failed_when` conditions are joined with an implicit `and`, meaning the task only fails when *all* conditions are met. If you want to trigger a failure when any of the conditions is met, you must define the conditions in a string with an explicit `or` operator.

You may check for failure by searching for a word or phrase in the output of a command

```yaml
- name: Fail task when the command error output prints FAILED
  ansible.builtin.command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
```

or based on the return code

```yaml
- name: Fail task when both files are identical
  ansible.builtin.raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

You can also combine multiple conditions for failure. This task will fail if both conditions are true:

```yaml
- name: Check if a file exists in temp and fail task if it does
  ansible.builtin.command: ls /tmp/this_should_not_be_here
  register: result
  failed_when:
    - result.rc == 0
    - '"No such" not in result.stdout'
```

If you want the task to fail when only one condition is satisfied, change the `failed_when` definition to

```yaml
failed_when: result.rc == 0 or "No such" not in result.stdout
```

If you have too many conditions to fit neatly into one line, you can split it into a multi-line YAML value with >.

```
- name: example of many failed_when conditions with OR
  ansible.builtin.shell: "./myBinary"
  register: ret
  failed_when: >
    ("No such file or directory" in ret.stdout) or
    (ret.stderr != '') or
    (ret.rc == 10)
```

Defining "changed"

Ansible lets you define when a particular task has "changed" a remote node using the `changed_when` conditional. This lets you determine, based on return codes or output, whether a change should be reported in Ansible statistics and whether a handler should be triggered or not. As with all conditionals in Ansible, lists of multiple `changed_when` conditions are joined with an implicit `and`, meaning the task only reports a change when *all* conditions are met. If you want to report a change when any of the conditions is met, you must define the conditions in a string with an explicit `or` operator. For example:

```
tasks:

  - name: Report 'changed' when the return code is not equal to 2
    ansible.builtin.shell: /usr/bin/billybass --mode="take me to the river"
    register: bass_result
    changed_when: "bass_result.rc != 2"

  - name: This will never report 'changed' status
    ansible.builtin.shell: wall 'beep'
    changed_when: False
```

You can also combine multiple conditions to override "changed" result.

```
- name: Combine multiple conditions to override 'changed' result
  ansible.builtin.command: /bin/fake_command
  register: result
  ignore_errors: True
  changed_when:
    - '"ERROR" in result.stderr'
    - result.rc == 2
```

Note: Just like `when` these two conditionals do not require templating delimiters (`{{ }}`) as they are implied.

See *Defining failure* for more conditional syntax examples.

Ensuring success for command and shell

The command and shell modules care about return codes, so if you have a command whose successful exit code is not zero, you can do this:

```
tasks:
  - name: Run this command and ignore the result
    ansible.builtin.shell: /usr/bin/somecommand || /bin/true
```

Aborting a play on all hosts

Sometimes you want a failure on a single host, or failures on a certain percentage of hosts, to abort the entire play on all hosts. You can stop play execution after the first failure happens with `any_errors_fatal`. For finer-grained control, you can use `max_fail_percentage` to abort the run after a given percentage of hosts has failed.

Aborting on the first error: any_errors_fatal

If you set `any_errors_fatal` and a task returns an error, Ansible finishes the fatal task on all hosts in the current batch, then stops executing the play on all hosts. Subsequent tasks and plays are not executed. You can recover from fatal errors by adding a *rescue section* to the block. You can set `any_errors_fatal` at the play or block level.

```
- hosts: somehosts
  any_errors_fatal: true
  roles:
    - myrole

- hosts: somehosts
  tasks:
    - block:
        - include_tasks: mytasks.yml
      any_errors_fatal: true
```

You can use this feature when all tasks must be 100% successful to continue playbook execution. For example, if you run a service on machines in multiple data centers with load balancers to pass traffic from users to the service, you want all load balancers to be disabled before you stop the service for maintenance. To ensure that any failure in the task that disables the load balancers will stop all other tasks:

```
---
- hosts: load_balancers_dc_a
  any_errors_fatal: true

  tasks:
    - name: Shut down datacenter 'A'
      ansible.builtin.command: /usr/bin/disable-dc

- hosts: frontends_dc_a

  tasks:
    - name: Stop service
      ansible.builtin.command: /usr/bin/stop-software
```

(continues on next page)

(continued from previous page)

```
    - name: Update software
      ansible.builtin.command: /usr/bin/upgrade-software

- hosts: load_balancers_dc_a

  tasks:
    - name: Start datacenter 'A'
      ansible.builtin.command: /usr/bin/enable-dc
```

In this example Ansible starts the software upgrade on the front ends only if all of the load balancers are successfully disabled.

Setting a maximum failure percentage

By default, Ansible continues to execute tasks as long as there are hosts that have not yet failed. In some situations, such as when executing a rolling update, you may want to abort the play when a certain threshold of failures has been reached. To achieve this, you can set a maximum failure percentage on a play:

```
---
- hosts: webservers
  max_fail_percentage: 30
  serial: 10
```

The `max_fail_percentage` setting applies to each batch when you use it with *serial*. In the example above, if more than 3 of the 10 servers in the first (or any) batch of servers failed, the rest of the play would be aborted.

Note: The percentage set must be exceeded, not equaled. For example, if serial were set to 4 and you wanted the task to abort the play when 2 of the systems failed, set the max_fail_percentage at 49 rather than 50.

Controlling errors in blocks

You can also use blocks to define responses to task errors. This approach is similar to exception handling in many programming languages. See *Handling errors with blocks* for details and examples.

See also:

Ansible playbooks
An introduction to playbooks

General tips
Tips and tricks for playbooks

Conditionals
Conditional statements in playbooks

Using Variables
All about variables

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Setting the remote environment

New in version 1.1.

You can use the environment keyword at the play, block, or task level to set an environment variable for an action on a remote host. With this keyword, you can enable using a proxy for a task that does http requests, set the required environment variables for language-specific version managers, and more.

When you set a value with environment: at the play or block level, it is available only to tasks within the play or block that are executed by the same user. The environment: keyword does not affect Ansible itself, Ansible configuration settings, the environment for other users, or the execution of other plugins like lookups and filters. Variables set with environment: do not automatically become Ansible facts, even when you set them at the play level. You must include an explicit gather_facts task in your playbook and set the environment keyword on that task to turn these values into Ansible facts.

Setting the remote environment in a task

You can set the environment directly at the task level.

```
- hosts: all
  remote_user: root

  tasks:

    - name: Install cobbler
      ansible.builtin.package:
        name: cobbler
        state: present
      environment:
        http_proxy: http://proxy.example.com:8080
```

You can re-use environment settings by defining them as variables in your play and accessing them in a task as you would access any stored Ansible variable.

```
- hosts: all
  remote_user: root

  # create a variable named "proxy_env" that is a dictionary
  vars:
    proxy_env:
      http_proxy: http://proxy.example.com:8080

  tasks:

    - name: Install cobbler
      ansible.builtin.package:
        name: cobbler
        state: present
      environment: "{{ proxy_env }}"
```

You can store environment settings for re-use in multiple playbooks by defining them in a group_vars file.

```
---
# file: group_vars/boston

ntp_server: ntp.bos.example.com
backup: bak.bos.example.com
proxy_env:
  http_proxy: http://proxy.bos.example.com:8080
  https_proxy: http://proxy.bos.example.com:8080
```

You can set the remote environment at the play level.

```
- hosts: testing

  roles:
     - php
     - nginx

  environment:
    http_proxy: http://proxy.example.com:8080
```

These examples show proxy settings, but you can provide any number of settings this way.

Working with language-specific version managers

Some language-specific version managers (such as rbenv and nvm) require you to set environment variables while these tools are in use. When using these tools manually, you usually source some environment variables from a script or from lines added to your shell configuration file. In Ansible, you can do this with the environment keyword at the play level.

```
---
### A playbook demonstrating a common npm workflow:
# - Check for package.json in the application directory
# - If package.json exists:
#   * Run npm prune
#   * Run npm install

- hosts: application
  become: false

  vars:
    node_app_dir: /var/local/my_node_app

  environment:
    NVM_DIR: /var/local/nvm
    PATH: /var/local/nvm/versions/node/v4.2.1/bin:{{ ansible_env.PATH }}

  tasks:
  - name: Check for package.json
    ansible.builtin.stat:
      path: '{{ node_app_dir }}/package.json'
    register: packagejson

  - name: Run npm prune
```

(continues on next page)

(continued from previous page)

```
    ansible.builtin.command: npm prune
    args:
      chdir: '{{ node_app_dir }}'
    when: packagejson.stat.exists

  - name: Run npm install
    community.general.npm:
      path: '{{ node_app_dir }}'
    when: packagejson.stat.exists
```

Note: The example above uses `ansible_env` as part of the PATH. Basing variables on `ansible_env` is risky. Ansible populates `ansible_env` values by gathering facts, so the value of the variables depends on the remote_user or become_user Ansible used when gathering those facts. If you change remote_user/become_user the values in `ansible_env` may not be the ones you expect.

Warning: Environment variables are normally passed in clear text (shell plugin dependent) so they are not a recommended way of passing secrets to the module being executed.

You can also specify the environment at the task level.

```
---
- name: Install ruby 2.3.1
  ansible.builtin.command: rbenv install {{ rbenv_ruby_version }}
  args:
    creates: '{{ rbenv_root }}/versions/{{ rbenv_ruby_version }}/bin/ruby'
  vars:
    rbenv_root: /usr/local/rbenv
    rbenv_ruby_version: 2.3.1
  environment:
    CONFIGURE_OPTS: '--disable-install-doc'
    RBENV_ROOT: '{{ rbenv_root }}'
    PATH: '{{ rbenv_root }}/bin:{{ rbenv_root }}/shims:{{ rbenv_plugins }}/ruby-build/
↪bin:{{ ansible_env.PATH }}'
```

See also:

Ansible playbooks
    An introduction to playbooks

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Re-using Ansible artifacts

You can write a simple playbook in one very large file, and most users learn the one-file approach first. However, breaking your automation work up into smaller files is an excellent way to organize complex sets of tasks and reuse them. Smaller, more distributed artifacts let you re-use the same variables, tasks, and plays in multiple playbooks to address different use cases. You can use distributed artifacts across multiple parent playbooks or even multiple times within one playbook. For example, you might want to update your customer database as part of several different playbooks. If you put all the tasks related to updating your database in a tasks file or a role, you can re-use them in many playbooks while only maintaining them in one place.

Creating re-usable files and roles

Ansible offers four distributed, re-usable artifacts: variables files, task files, playbooks, and roles.

- A variables file contains only variables.
- A task file contains only tasks.
- A playbook contains at least one play, and may contain variables, tasks, and other content. You can re-use tightly focused playbooks, but you can only re-use them statically, not dynamically.
- A role contains a set of related tasks, variables, defaults, handlers, and even modules or other plugins in a defined file-tree. Unlike variables files, task files, or playbooks, roles can be easily uploaded and shared through Ansible Galaxy. See *Roles* for details about creating and using roles.

New in version 2.4.

Re-using playbooks

You can incorporate multiple playbooks into a main playbook. However, you can only use imports to re-use playbooks. For example:

```
- import_playbook: webservers.yml
- import_playbook: databases.yml
```

Importing incorporates playbooks in other playbooks statically. Ansible runs the plays and tasks in each imported playbook in the order they are listed, just as if they had been defined directly in the main playbook.

You can select which playbook you want to import at runtime by defining your imported playbook filename with a variable, then passing the variable with either `--extra-vars` or the `vars` keyword. For example:

```
- import_playbook: "/path/to/{{ import_from_extra_var }}"
- import_playbook: "{{ import_from_vars }}"
  vars:
    import_from_vars: /path/to/one_playbook.yml
```

If you run this playbook with `ansible-playbook my_playbook -e import_from_extra_var=other_playbook.yml`, Ansible imports both one_playbook.yml and other_playbook.yml.

When to turn a playbook into a role

For some use cases, simple playbooks work well. However, starting at a certain level of complexity, roles work better than playbooks. A role lets you store your defaults, handlers, variables, and tasks in separate directories, instead of in a single long document. Roles are easy to share on Ansible Galaxy. For complex use cases, most users find roles easier to read, understand, and maintain than all-in-one playbooks.

Re-using files and roles

Ansible offers two ways to re-use files and roles in a playbook: dynamic and static.

- For dynamic re-use, add an `include_*` task in the tasks section of a play:
  - include_role
  - include_tasks
  - include_vars
- For static re-use, add an `import_*` task in the tasks section of a play:
  - import_role
  - import_tasks

Task include and import statements can be used at arbitrary depth.

You can still use the bare *roles* keyword at the play level to incorporate a role in a playbook statically. However, the bare include keyword, once used for both task files and playbook-level includes, is now deprecated.

Includes: dynamic re-use

Including roles, tasks, or variables adds them to a playbook dynamically. Ansible processes included files and roles as they come up in a playbook, so included tasks can be affected by the results of earlier tasks within the top-level playbook. Included roles and tasks are similar to handlers - they may or may not run, depending on the results of other tasks in the top-level playbook.

The primary advantage of using `include_*` statements is looping. When a loop is used with an include, the included tasks or role will be executed once for each item in the loop.

The filenames for included roles, tasks, and vars are templated before inclusion.

You can pass variables into includes. See *Variable precedence: Where should I put a variable?* for more details on variable inheritance and precedence.

Imports: static re-use

Importing roles, tasks, or playbooks adds them to a playbook statically. Ansible pre-processes imported files and roles before it runs any tasks in a playbook, so imported content is never affected by other tasks within the top-level playbook.

The filenames for imported roles and tasks support templating, but the variables must be available when Ansible is pre-processing the imports. This can be done with the `vars` keyword or by using `--extra-vars`.

You can pass variables to imports. You must pass variables if you want to run an imported file more than once in a playbook. For example:

```yaml
tasks:
- import_tasks: wordpress.yml
  vars:
    wp_user: timmy

- import_tasks: wordpress.yml
  vars:
    wp_user: alice

- import_tasks: wordpress.yml
  vars:
    wp_user: bob
```

See *Variable precedence: Where should I put a variable?* for more details on variable inheritance and precedence.

Comparing includes and imports: dynamic and static re-use

Each approach to re-using distributed Ansible artifacts has advantages and limitations. You may choose dynamic re-use for some playbooks and static re-use for others. Although you can use both dynamic and static re-use in a single playbook, it is best to select one approach per playbook. Mixing static and dynamic re-use can introduce difficult-to-diagnose bugs into your playbooks. This table summarizes the main differences so you can choose the best approach for each playbook you create.

	Include_*	Import_*
Type of re-use	Dynamic	Static
When processed	At runtime, when encountered	Pre-processed during playbook parsing
Task or play	All includes are tasks	`import_playbook` cannot be a task
Task options	Apply only to include task itself	Apply to all child tasks in import
Calling from loops	Executed once for each loop item	Cannot be used in a loop
Using `--list-tags`	Tags within includes not listed	All tags appear with `--list-tags`
Using `--list-tasks`	Tasks within includes not listed	All tasks appear with `--list-tasks`
Notifying handlers	Cannot trigger handlers within includes	Can trigger individual imported handlers
Using `--start-at-task`	Cannot start at tasks within includes	Can start at imported tasks
Using inventory variables	Can `include_*: {{ inventory_var }}`	Cannot `import_*: {{ inventory_var }}`
With playbooks	No `include_playbook`	Can import full playbooks
With variables files	Can include variables files	Use `vars_files:` to import variables

Note:

- There are also big differences in resource consumption and performance, imports are quite lean and fast, while includes require a lot of management and accounting.

Re-using tasks as handlers

You can also use includes and imports in the *Handlers: running operations on change* section of a playbook. For instance, if you want to define how to restart Apache, you only have to do that once for all of your playbooks. You might make a `restarts.yml` file that looks like:

```
# restarts.yml
- name: Restart apache
  ansible.builtin.service:
    name: apache
    state: restarted

- name: Restart mysql
  ansible.builtin.service:
    name: mysql
    state: restarted
```

You can trigger handlers from either an import or an include, but the procedure is different for each method of re-use. If you include the file, you must notify the include itself, which triggers all the tasks in `restarts.yml`. If you import the file, you must notify the individual task(s) within `restarts.yml`. You can mix direct tasks and handlers with included or imported tasks and handlers.

Triggering included (dynamic) handlers

Includes are executed at run-time, so the name of the include exists during play execution, but the included tasks do not exist until the include itself is triggered. To use the `Restart apache` task with dynamic re-use, refer to the name of the include itself. This approach triggers all tasks in the included file as handlers. For example, with the task file shown above:

```
- name: Trigger an included (dynamic) handler
  hosts: localhost
  handlers:
    - name: Restart services
      include_tasks: restarts.yml
  tasks:
    - command: "true"
      notify: Restart services
```

Triggering imported (static) handlers

Imports are processed before the play begins, so the name of the import no longer exists during play execution, but the names of the individual imported tasks do exist. To use the `Restart apache` task with static re-use, refer to the name of each task or tasks within the imported file. For example, with the task file shown above:

```
- name: Trigger an imported (static) handler
  hosts: localhost
  handlers:
    - name: Restart services
      import_tasks: restarts.yml
  tasks:
    - command: "true"
      notify: Restart apache
    - command: "true"
      notify: Restart mysql
```

See also:

Utilities modules
Documentation of the `include*` and `import*` modules discussed here.

Working with playbooks
Review the basic Playbook language features

Using Variables
All about variables in playbooks

Conditionals
Conditionals in playbooks

Loops
Loops in playbooks

General tips
Tips and tricks for playbooks

Galaxy User Guide
How to share roles on galaxy, role management

GitHub Ansible examples
Complete playbook files from the GitHub project source

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Roles

Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure. After you group your content in roles, you can easily reuse them and share them with other users.

Role directory structure

An Ansible role has a defined directory structure with eight main standard directories. You must include at least one of these directories in each role. You can omit any directories the role does not use. For example:

```
# playbooks
site.yml
webservers.yml
fooservers.yml
```

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the␣
↪webtier role
```

(continues on next page)

(continued from previous page)

```
    monitoring/           # ""
    fooapp/               # ""
```

By default Ansible will look in each directory within a role for a `main.yml` file for relevant content (also `main.yaml` and `main`):

- `tasks/main.yml` - the main list of tasks that the role executes.
- `handlers/main.yml` - handlers, which may be used within or outside this role.
- `library/my_module.py` - modules, which may be used within this role (see *Embedding modules and plugins in roles* for more information).
- `defaults/main.yml` - default variables for the role (see *Using Variables* for more information). These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.
- `vars/main.yml` - other variables for the role (see *Using Variables* for more information).
- `files/main.yml` - files that the role deploys.
- `templates/main.yml` - templates that the role deploys.
- `meta/main.yml` - metadata for the role, including role dependencies and optional Galaxy metadata such as platforms supported.

You can add other YAML files in some directories. For example, you can place platform-specific tasks in separate files and refer to them in the `tasks/main.yml` file:

```
# roles/example/tasks/main.yml
- name: Install the correct web server for RHEL
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'

- name: Install the correct web server for Debian
  import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'

# roles/example/tasks/redhat.yml
- name: Install web server
  ansible.builtin.yum:
    name: "httpd"
    state: present

# roles/example/tasks/debian.yml
- name: Install web server
  ansible.builtin.apt:
    name: "apache2"
    state: present
```

Roles may also include modules and other plugin types in a directory called `library`. For more information, please refer to *Embedding modules and plugins in roles* below.

Storing and finding roles

By default, Ansible looks for roles in the following locations:

- in collections, if you are using them
- in a directory called `roles/`, relative to the playbook file
- in the configured *roles_path*. The default search path is `~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`.
- in the directory where the playbook file is located

If you store your roles in a different location, set the *roles_path* configuration option so Ansible can find your roles. Checking shared roles into a single location makes them easier to use in multiple playbooks. See *Configuring Ansible* for details about managing settings in ansible.cfg.

Alternatively, you can call a role with a fully qualified path:

```
---
- hosts: webservers
  roles:
    - role: '/path/to/my/roles/common'
```

Using roles

You can use roles in three ways:

- at the play level with the `roles` option: This is the classic way of using roles in a play.
- at the tasks level with `include_role`: You can reuse roles dynamically anywhere in the `tasks` section of a play using `include_role`.
- at the tasks level with `import_role`: You can reuse roles statically anywhere in the `tasks` section of a play using `import_role`.

Using roles at the play level

The classic (original) way to use roles is with the `roles` option for a given play:

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

When you use the `roles` option at the play level, for each role 'x':

- If roles/x/tasks/main.yml exists, Ansible adds the tasks in that file to the play.
- If roles/x/handlers/main.yml exists, Ansible adds the handlers in that file to the play.
- If roles/x/vars/main.yml exists, Ansible adds the variables in that file to the play.
- If roles/x/defaults/main.yml exists, Ansible adds the variables in that file to the play.
- If roles/x/meta/main.yml exists, Ansible adds any role dependencies in that file to the list of roles.

- Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely.

When you use the `roles` option at the play level, Ansible treats the roles as static imports and processes them during playbook parsing. Ansible executes each play in this order:

- Any `pre_tasks` defined in the play.
- Any handlers triggered by pre_tasks.
- Each role listed in `roles:`, in the order listed. Any role dependencies defined in the role's `meta/main.yml` run first, subject to tag filtering and conditionals. See *Using role dependencies* for more details.
- Any `tasks` defined in the play.
- Any handlers triggered by the roles or tasks.
- Any `post_tasks` defined in the play.
- Any handlers triggered by post_tasks.

Note: If using tags with tasks in a role, be sure to also tag your pre_tasks, post_tasks, and role dependencies and pass those along as well, especially if the pre/post tasks and role dependencies are used for monitoring outage window control or load balancing. See *Tags* for details on adding and using tags.

You can pass other keywords to the `roles` option:

```
---
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
      tags: typeB
```

When you add a tag to the `role` option, Ansible applies the tag to ALL tasks within the role.

When using `vars:` within the `roles:` section of a playbook, the variables are added to the play variables, making them available to all tasks within the play before and after the role. This behavior can be changed by *DEFAULT_PRIVATE_ROLE_VARS*.

Including roles: dynamic reuse

You can reuse roles dynamically anywhere in the `tasks` section of a play using `include_role`. While roles added in a `roles` section run before any other tasks in a play, included roles run in the order they are defined. If there are other tasks before an `include_role` task, the other tasks will run first.

To include a role:

```
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs before the example role"

    - name: Include the example role
      include_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs after the example role"
```

You can pass other keywords, including variables and tags, when including roles:

```
---
- hosts: webservers
  tasks:
    - name: Include the foo_app_instance role
      include_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
  ...
```

When you add a *tag* to an `include_role` task, Ansible applies the tag *only* to the include itself. This means you can pass `--tags` to run only selected tasks from the role, if those tasks themselves have the same tag as the include statement. See *Selectively running tagged tasks in re-usable files* for details.

You can conditionally include a role:

```
---
- hosts: webservers
  tasks:
    - name: Include the some_role role
      include_role:
        name: some_role
      when: "ansible_facts['os_family'] == 'RedHat'"
```

Importing roles: static reuse

You can reuse roles statically anywhere in the `tasks` section of a play using `import_role`. The behavior is the same as using the `roles` keyword. For example:

```yaml
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "before we run our role"

    - name: Import the example role
      import_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "after we ran our role"
```

You can pass other keywords, including variables and tags, when importing roles:

```yaml
---
- hosts: webservers
  tasks:
    - name: Import the foo_app_instance role
      import_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
  ...
```

When you add a tag to an `import_role` statement, Ansible applies the tag to *all* tasks within the role. See *Tag inheritance: adding tags to multiple tasks* for details.

Role argument validation

Beginning with version 2.11, you may choose to enable role argument validation based on an argument specification. This specification is defined in the `meta/argument_specs.yml` file (or with the `.yaml` file extension). When this argument specification is defined, a new task is inserted at the beginning of role execution that will validate the parameters supplied for the role against the specification. If the parameters fail validation, the role will fail execution.

Note: Ansible also supports role specifications defined in the role `meta/main.yml` file, as well. However, any role that defines the specs within this file will not work on versions below 2.11. For this reason, we recommend using the `meta/argument_specs.yml` file to maintain backward compatibility.

Note: When role argument validation is used on a role that has defined *dependencies*, then validation on those dependencies will run before the dependent role, even if argument validation fails for the dependent role.

Specification format

The role argument specification must be defined in a top-level `argument_specs` block within the role `meta/argument_specs.yml` file. All fields are lower-case.

entry-point-name

- The name of the role entry point.
- This should be `main` in the case of an unspecified entry point.
- This will be the base name of the tasks file to execute, with no `.yml` or `.yaml` file extension.

  short_description

  - A short, one-line description of the entry point.
  - The `short_description` is displayed by `ansible-doc -t role -l`.

  description

  - A longer description that may contain multiple lines.

  author

  - Name of the entry point authors.
  - Use a multi-line list if there is more than one author.

  options

  - Options are often called "parameters" or "arguments". This section defines those options.
  - For each role option (argument), you may include:

    option-name

    - The name of the option/argument.

    description

    - Detailed explanation of what this option does. It should be written in full sentences.

    type

    - The data type of the option. See *Argument spec* for allowed values for `type`. Default is `str`.
    - If an option is of type `list`, `elements` should be specified.

    required

    - Only needed if `true`.
    - If missing, the option is not required.

    default

    - If `required` is false/missing, `default` may be specified (assumed 'null' if missing).
    - Ensure that the default value in the docs matches the default value in the code. The actual default for the role variable will always come from `defaults/main.yml`.

- The default field must not be listed as part of the description, unless it requires additional information or conditions.
- If the option is a boolean value, you should use *true/false* if you want to be compatible with *ansible-lint*.

choices

- List of option values.
- Should be absent if empty.

elements

- Specifies the data type for list elements when type is `list`.

options

- If this option takes a dict or list of dicts, you can define the structure here.

Sample specification

```
# roles/myapp/meta/argument_specs.yml
---
argument_specs:
  # roles/myapp/tasks/main.yml entry point
  main:
    short_description: The main entry point for the myapp role.
    options:
      myapp_int:
        type: "int"
        required: false
        default: 42
        description: "The integer value, defaulting to 42."

      myapp_str:
        type: "str"
        required: true
        description: "The string value"

  # roles/myapp/tasks/alternate.yml entry point
  alternate:
    short_description: The alternate entry point for the myapp role.
    options:
      myapp_int:
        type: "int"
        required: false
        default: 1024
        description: "The integer value, defaulting to 1024."
```

Running a role multiple times in one play

Ansible only executes each role once in a play, even if you define it multiple times, unless the parameters defined on the role are different for each definition. For example, Ansible only runs the role `foo` once in a play like this:

```
---
- hosts: webservers
  roles:
    - foo
    - bar
    - foo
```

You have two options to force Ansible to run a role more than once.

Passing different parameters

If you pass different parameters in each role definition, Ansible runs the role more than once. Providing different variable values is not the same as passing different role parameters. You must use the `roles` keyword for this behavior, since `import_role` and `include_role` do not accept role parameters.

This play runs the `foo` role twice:

```
---
- hosts: webservers
  roles:
    - { role: foo, message: "first" }
    - { role: foo, message: "second" }
```

This syntax also runs the `foo` role twice;

```
---
- hosts: webservers
  roles:
    - role: foo
      message: "first"
    - role: foo
      message: "second"
```

In these examples, Ansible runs `foo` twice because each role definition has different parameters.

Using `allow_duplicates: true`

Add `allow_duplicates: true` to the `meta/main.yml` file for the role:

```
# playbook.yml
---
- hosts: webservers
  roles:
    - foo
    - foo

# roles/foo/meta/main.yml
```

(continues on next page)

(continued from previous page)

```
---
allow_duplicates: true
```

In this example, Ansible runs `foo` twice because we have explicitly enabled it to do so.

Using role dependencies

Role dependencies let you automatically pull in other roles when using a role.

Role dependencies are prerequisites, not true dependencies. The roles do not have a parent/child relationship. Ansible loads all listed roles, runs the roles listed under `dependencies` first, then runs the role that lists them. The play object is the parent of all roles, including roles called by a `dependencies` list.

Role dependencies are stored in the `meta/main.yml` file within the role directory. This file should contain a list of roles and parameters to insert before the specified role. For example:

```
# roles/myapp/meta/main.yml
---
dependencies:
  - role: common
    vars:
      some_parameter: 3
  - role: apache
    vars:
      apache_port: 80
  - role: postgres
    vars:
      dbname: blarg
      other_parameter: 12
```

Ansible always executes roles listed in `dependencies` before the role that lists them. Ansible executes this pattern recursively when you use the `roles` keyword. For example, if you list role `foo` under `roles:`, role `foo` lists role `bar` under `dependencies` in its meta/main.yml file, and role `bar` lists role `baz` under `dependencies` in its meta/main.yml, Ansible executes `baz`, then `bar`, then `foo`.

Running role dependencies multiple times in one play

Ansible treats duplicate role dependencies like duplicate roles listed under `roles:`: Ansible only executes role dependencies once, even if defined multiple times, unless the parameters, tags, or when clause defined on the role are different for each definition. If two roles in a play both list a third role as a dependency, Ansible only runs that role dependency once, unless you pass different parameters, tags, when clause, or use `allow_duplicates: true` in the role you want to run multiple times. See *Galaxy role dependencies* for more details.

Note: Role deduplication does not consult the invocation signature of parent roles. Additionally, when using `vars:` instead of role params, there is a side effect of changing variable scoping. Using `vars:` results in those variables being scoped at the play level. In the below example, using `vars:` would cause `n` to be defined as 4 through the entire play, including roles called before it.

In addition to the above, users should be aware that role de-duplication occurs before variable evaluation. This means that *Lazy Evaluation* may make seemingly different role invocations equivalently the same, preventing the role from running more than once.

For example, a role named `car` depends on a role named `wheel` as follows:

```yaml
---
dependencies:
  - role: wheel
    n: 1
  - role: wheel
    n: 2
  - role: wheel
    n: 3
  - role: wheel
    n: 4
```

And the `wheel` role depends on two roles: `tire` and `brake`. The `meta/main.yml` for wheel would then contain the following:

```yaml
---
dependencies:
  - role: tire
  - role: brake
```

And the `meta/main.yml` for `tire` and `brake` would contain the following:

```yaml
---
allow_duplicates: true
```

The resulting order of execution would be as follows:

```
tire(n=1)
brake(n=1)
wheel(n=1)
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

To use `allow_duplicates: true` with role dependencies, you must specify it for the role listed under `dependencies`, not for the role that lists it. In the example above, `allow_duplicates: true` appears in the `meta/main.yml` of the `tire` and `brake` roles. The `wheel` role does not require `allow_duplicates: true`, because each instance defined by `car` uses different parameter values.

Note: See *Using Variables* for details on how Ansible chooses among variable values defined in different places (variable inheritance and scope). Also deduplication happens ONLY at the play level, so multiple plays in the same playbook may rerun the roles.

Embedding modules and plugins in roles

Note: This applies only to standalone roles. Roles in collections do not support plugin embedding; they must use the collection's `plugins` structure to distribute plugins.

If you write a custom module (see *Should you develop a module?*) or a plugin (see *Developing plugins*), you might wish to distribute it as part of a role. For example, if you write a module that helps configure your company's internal software, and you want other people in your organization to use this module, but you do not want to tell everyone how to configure their Ansible library path, you can include the module in your internal_config role.

To add a module or a plugin to a role: Alongside the 'tasks' and 'handlers' structure of a role, add a directory named 'library' and then include the module directly inside the 'library' directory.

Assuming you had this:

```
roles/
    my_custom_modules/
        library/
            module1
            module2
```

The module will be usable in the role itself, as well as any roles that are called *after* this role, as follows:

```
---
- hosts: webservers
  roles:
    - my_custom_modules
    - some_other_role_using_my_custom_modules
    - yet_another_role_using_my_custom_modules
```

If necessary, you can also embed a module in a role to modify a module in Ansible's core distribution. For example, you can use the development version of a particular module before it is released in production releases by copying the module and embedding the copy in a role. Use this approach with caution, as API signatures may change in core components, and this workaround is not guaranteed to work.

The same mechanism can be used to embed and distribute plugins in a role, using the same schema. For example, for a filter plugin:

```
roles/
    my_custom_filter/
        filter_plugins
            filter1
            filter2
```

These filters can then be used in a Jinja template in any role called after 'my_custom_filter'.

Sharing roles: Ansible Galaxy

Ansible Galaxy is a free site for finding, downloading, rating, and reviewing all kinds of community-developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

The client `ansible-galaxy` is included in Ansible. The Galaxy client allows you to download roles from Ansible Galaxy and provides an excellent default framework for creating your own roles.

Read the Ansible Galaxy documentation page for more information. A page that refers back to this one frequently is the Galaxy Roles document which explains the required metadata your role needs for use in Galaxy <https://galaxy.ansible.com/docs/contributing/creating_role.html>.

See also:

Galaxy User Guide
    How to create new roles, share roles on Galaxy, role management

YAML Syntax
    Learn about YAML syntax

Working with playbooks
    Review the basic Playbook language features

General tips
    Tips and tricks for playbooks

Using Variables
    Variables in playbooks

Conditionals
    Conditionals in playbooks

Loops
    Loops in playbooks

Tags
    Using tags to select or skip roles/tasks in long playbooks

Collection Index
    Browse existing collections, modules, and plugins

Should you develop a module?
    Extending Ansible by writing your own modules

GitHub Ansible examples
    Complete playbook files from the GitHub project source

Mailing List
    Questions? Help? Ideas? Stop by the list on Google Groups

Module defaults

If you frequently call the same module with the same arguments, it can be useful to define default arguments for that particular module using the `module_defaults` keyword.

Here is a basic example:

```
- hosts: localhost
  module_defaults:
    ansible.builtin.file:
```

(continues on next page)

(continued from previous page)

```
      owner: root
      group: root
      mode: 0755
  tasks:
    - name: Create file1
      ansible.builtin.file:
        state: touch
        path: /tmp/file1

    - name: Create file2
      ansible.builtin.file:
        state: touch
        path: /tmp/file2

    - name: Create file3
      ansible.builtin.file:
        state: touch
        path: /tmp/file3
```

The `module_defaults` keyword can be used at the play, block, and task level. Any module arguments explicitly specified in a task will override any established default for that module argument.

```
- block:
    - name: Print a message
      ansible.builtin.debug:
        msg: "Different message"
  module_defaults:
    ansible.builtin.debug:
      msg: "Default message"
```

You can remove any previously established defaults for a module by specifying an empty dict.

```
- name: Create file1
  ansible.builtin.file:
    state: touch
    path: /tmp/file1
  module_defaults:
    file: {}
```

Note: Any module defaults set at the play level (and block/task level when using `include_role` or `import_role`) will apply to any roles used, which may cause unexpected behavior in the role.

Here are some more realistic use cases for this feature.

Interacting with an API that requires auth.

```
- hosts: localhost
  module_defaults:
    ansible.builtin.uri:
      force_basic_auth: true
      user: some_user
      password: some_password
```

(continues on next page)

(continued from previous page)

```
  tasks:
    - name: Interact with a web service
      ansible.builtin.uri:
        url: http://some.api.host/v1/whatever1

    - name: Interact with a web service
      ansible.builtin.uri:
        url: http://some.api.host/v1/whatever2

    - name: Interact with a web service
      ansible.builtin.uri:
        url: http://some.api.host/v1/whatever3
```

Setting a default AWS region for specific EC2-related modules.

```
- hosts: localhost
  vars:
    my_region: us-west-2
  module_defaults:
    amazon.aws.ec2:
      region: '{{ my_region }}'
    community.aws.ec2_instance_info:
      region: '{{ my_region }}'
    amazon.aws.ec2_vpc_net_info:
      region: '{{ my_region }}'
```

Module defaults groups

New in version 2.7.

Ansible 2.7 adds a preview-status feature to group together modules that share common sets of parameters. This makes it easier to author playbooks making heavy use of API-based modules such as cloud modules.

Group	Purpose	Ansible Version
aws	Amazon Web Services	2.7
azure	Azure	2.7
gcp	Google Cloud Platform	2.7
k8s	Kubernetes	2.8
os	OpenStack	2.8
acme	ACME	2.10
docker*	Docker	2.10
ovirt	oVirt	2.10
vmware	VMware	2.10

- The docker_stack module is not included in the `docker` defaults group.

Use the groups with `module_defaults` by prefixing the group name with `group/` - for example `group/aws`.

In a playbook, you can set module defaults for whole groups of modules, such as setting a common AWS region.

```
# example_play.yml
- hosts: localhost
```

(continues on next page)

(continued from previous page)

```yaml
  module_defaults:
    group/aws:
      region: us-west-2
  tasks:
  - name: Get info
    aws_s3_bucket_info:

  # now the region is shared between both info modules

  - name: Get info
    ec2_ami_info:
      filters:
        name: 'RHEL*7.5*'
```

In ansible-core 2.12, collections can define their own groups in the `meta/runtime.yml` file. `module_defaults` does not take the `collections` keyword into account, so the fully qualified group name must be used for new groups in `module_defaults`.

Here is an example `runtime.yml` file for a collection and a sample playbook using the group.

```yaml
# collections/ansible_collections/ns/coll/meta/runtime.yml
action_groups:
  groupname:
    - module
    - another.collection.module
```

```yaml
- hosts: localhost
  module_defaults:
    group/ns.coll.groupname:
      option_name: option_value
  tasks:
    - ns.coll.module:
    - another.collection.module
```

Interactive input: prompts

If you want your playbook to prompt the user for certain input, add a 'vars_prompt' section. Prompting the user for variables lets you avoid recording sensitive data like passwords. In addition to security, prompts support flexibility. For example, if you use one playbook across multiple software releases, you could prompt for the particular release version.

- *Hashing values supplied by `vars_prompt`*
- *Allowing special characters in `vars_prompt` values*

Here is a most basic example:

```yaml
---
- hosts: all
  vars_prompt:

```

(continues on next page)

(continued from previous page)

```
    - name: username
      prompt: What is your username?
      private: false

    - name: password
      prompt: What is your password?

  tasks:

    - name: Print a message
      ansible.builtin.debug:
        msg: 'Logging in as {{ username }}'
```

The user input is hidden by default but it can be made visible by setting `private: no`.

Note: Prompts for individual `vars_prompt` variables will be skipped for any variable that is already defined through the command line `--extra-vars` option, or when running from a non-interactive session (such as cron or Ansible AWX). See *Defining variables at runtime*.

If you have a variable that changes infrequently, you can provide a default value that can be overridden.

```
vars_prompt:

  - name: release_version
    prompt: Product release version
    default: "1.0"
```

Hashing values supplied by `vars_prompt`

You can hash the entered value so you can use it, for instance, with the user module to define a password:

```
vars_prompt:

  - name: my_password2
    prompt: Enter password2
    private: true
    encrypt: sha512_crypt
    confirm: true
    salt_size: 7
```

If you have Passlib installed, you can use any crypt scheme the library supports:

- *des_crypt* - DES Crypt
- *bsdi_crypt* - BSDi Crypt
- *bigcrypt* - BigCrypt
- *crypt16* - Crypt16
- *md5_crypt* - MD5 Crypt
- *bcrypt* - BCrypt

- *sha1_crypt* - SHA-1 Crypt
- *sun_md5_crypt* - Sun MD5 Crypt
- *sha256_crypt* - SHA-256 Crypt
- *sha512_crypt* - SHA-512 Crypt
- *apr_md5_crypt* - Apache's MD5-Crypt variant
- *phpass* - PHPass' Portable Hash
- *pbkdf2_digest* - Generic PBKDF2 Hashes
- *cta_pbkdf2_sha1* - Cryptacular's PBKDF2 hash
- *dlitz_pbkdf2_sha1* - Dwayne Litzenberger's PBKDF2 hash
- *scram* - SCRAM Hash
- *bsd_nthash* - FreeBSD's MCF-compatible nthash encoding

The only parameters accepted are 'salt' or 'salt_size'. You can use your own salt by defining 'salt', or have one generated automatically using 'salt_size'. By default Ansible generates a salt of size 8.

New in version 2.7.

If you do not have Passlib installed, Ansible uses the crypt library as a fallback. Ansible supports at most four crypt schemes, depending on your platform at most the following crypt schemes are supported:

- *bcrypt* - BCrypt
- *md5_crypt* - MD5 Crypt
- *sha256_crypt* - SHA-256 Crypt
- *sha512_crypt* - SHA-512 Crypt

New in version 2.8.

Allowing special characters in `vars_prompt` values

Some special characters, such as `{` and `%` can create templating errors. If you need to accept special characters, use the `unsafe` option:

```
vars_prompt:
  - name: my_password_with_weird_chars
    prompt: Enter password
    unsafe: true
    private: true
```

See also:

Ansible playbooks
An introduction to playbooks

Conditionals
Conditional statements in playbooks

Using Variables
All about variables

User Mailing List
Have a question? Stop by the google group!

Real-time chat
  How to join Ansible chat channels

Using Variables

Ansible uses variables to manage differences between systems. With Ansible, you can execute tasks and playbooks on multiple different systems with a single command. To represent the variations among those different systems, you can create variables with standard YAML syntax, including lists and dictionaries. You can define these variables in your playbooks, in your *inventory*, in re-usable *files* or *roles*, or at the command line. You can also create variables during a playbook run by registering the return value or values of a task as a new variable.

After you create variables, either by defining them in a file, passing them at the command line, or registering the return value or values of a task as a new variable, you can use those variables in module arguments, in *conditional "when" statements*, in *templates*, and in *loops*. The ansible-examples github repository contains many examples of using variables in Ansible.

Once you understand the concepts and examples on this page, read about *Ansible facts*, which are variables you retrieve from remote systems.

Creating valid variable names

Not all strings are valid Ansible variable names. A variable name can only include letters, numbers, and underscores. Python keywords or *playbook keywords* are not valid variable names. A variable name cannot begin with a number.

Variable names can begin with an underscore. In many programming languages, variables that begin with an underscore are private. This is not true in Ansible. Variables that begin with an underscore are treated exactly the same as any other variable. Do not rely on this convention for privacy or security.

This table gives examples of valid and invalid variable names:

Valid variable names	Not valid
`foo`	`*foo`, Python keywords such as `async` and `lambda`
`foo_env`	*playbook keywords* such as `environment`
`foo_port`	`foo-port`, `foo port`, `foo.port`
`foo5`, `_foo`	`5foo`, `12`

Simple variables

Simple variables combine a variable name with a single value. You can use this syntax (and the syntax for lists and dictionaries shown below) in a variety of places. For details about setting variables in inventory, in playbooks, in reusable files, in roles, or at the command line, see *Where to set variables*.

Defining simple variables

You can define a simple variable using standard YAML syntax. For example:

```
remote_install_path: /opt/my_app_config
```

Referencing simple variables

After you define a variable, use Jinja2 syntax to reference it. Jinja2 variables use double curly braces. For example, the expression `My amp goes to {{ max_amp_value }}` demonstrates the most basic form of variable substitution. You can use Jinja2 syntax in playbooks. For example:

```
ansible.builtin.template:
  src: foo.cfg.j2
  dest: '{{ remote_install_path }}/foo.cfg'
```

In this example, the variable defines the location of a file, which can vary from one system to another.

Note: Ansible allows Jinja2 loops and conditionals in *templates* but not in playbooks. You cannot create a loop of tasks. Ansible playbooks are pure machine-parseable YAML.

When to quote variables (a YAML gotcha)

If you start a value with `{{ foo }}`, you must quote the whole expression to create valid YAML syntax. If you do not quote the whole expression, the YAML parser cannot interpret the syntax - it might be a variable or it might be the start of a YAML dictionary. For guidance on writing YAML, see the *YAML Syntax* documentation.

If you use a variable without quotes like this:

```
- hosts: app_servers
  vars:
      app_path: {{ base_path }}/22
```

You will see: `ERROR! Syntax Error while loading YAML.` If you add quotes, Ansible works correctly:

```
- hosts: app_servers
  vars:
       app_path: "{{ base_path }}/22"
```

Boolean variables

Ansible accepts a broad range of values for boolean variables: `true/false`, `1/0`, `yes/no`, `True/False` and so on. The matching of valid strings is case insensitive. While documentation examples focus on `true/false` to be compatible with `ansible-lint` default settings, you can use any of the following:

Valid values	Description
`True` , `'true'` , `'t'` , `'yes'` , `'y'` , `'on'` , `'1'` , `1` , `1.0`	Truthy values
`False` , `'false'` , `'f'` , `'no'` , `'n'` , `'off'` , `'0'` , `0` , `0.0`	Falsy values

List variables

A list variable combines a variable name with multiple values. The multiple values can be stored as an itemized list or in square brackets `[]`, separated with commas.

Defining variables as lists

You can define variables with multiple values using YAML lists. For example:

```
region:
  - northeast
  - southeast
  - midwest
```

Referencing list variables

When you use variables defined as a list (also called an array), you can use individual, specific fields from that list. The first item in a list is item 0, the second item is item 1. For example:

```
region: "{{ region[0] }}"
```

The value of this expression would be "northeast".

Dictionary variables

A dictionary stores the data in key-value pairs. Usually, dictionaries are used to store related data, such as the information contained in an ID or a user profile.

Defining variables as key:value dictionaries

You can define more complex variables using YAML dictionaries. A YAML dictionary maps keys to values. For example:

```
foo:
  field1: one
  field2: two
```

Referencing key:value dictionary variables

When you use variables defined as a key:value dictionary (also called a hash), you can use individual, specific fields from that dictionary using either bracket notation or dot notation:

```
foo['field1']
foo.field1
```

Both of these examples reference the same value ("one"). Bracket notation always works. Dot notation can cause problems because some keys collide with attributes and methods of python dictionaries. Use bracket notation if you use keys which start and end with two underscores (which are reserved for special meanings in python) or are any of the known public attributes:

`add`, `append`, `as_integer_ratio`, `bit_length`, `capitalize`, `center`, `clear`, `conjugate`, `copy`, `count`, `decode`, `denominator`, `difference`, `difference_update`, `discard`, `encode`, `endswith`, `expandtabs`, `extend`, `find`, `format`, `fromhex`, `fromkeys`, `get`, `has_key`, `hex`, `imag`, `index`, `insert`, `intersection`, `intersection_update`, `isalnum`, `isalpha`, `isdecimal`, `isdigit`, `isdisjoint`, `is_integer`, `islower`, `isnumeric`, `isspace`, `issubset`, `issuperset`, `istitle`, `isupper`, `items`, `iteritems`, `iterkeys`, `itervalues`, `join`, `keys`, `ljust`, `lower`, `lstrip`, `numerator`, `partition`, `pop`, `popitem`, `real`, `remove`, `replace`, `reverse`, `rfind`, `rindex`, `rjust`, `rpartition`, `rsplit`, `rstrip`, `setdefault`, `sort`, `split`, `splitlines`, `startswith`, `strip`, `swapcase`, `symmetric_difference`, `symmetric_difference_update`, `title`, `translate`, `union`, `update`, `upper`, `values`, `viewitems`, `viewkeys`, `viewvalues`, `zfill`.

Registering variables

You can create variables from the output of an Ansible task with the task keyword `register`. You can use registered variables in any later tasks in your play. For example:

```
- hosts: web_servers

  tasks:

     - name: Run a shell command and register its output as a variable
       ansible.builtin.shell: /usr/bin/foo
       register: foo_result
       ignore_errors: true

     - name: Run a shell command using output of the previous task
       ansible.builtin.shell: /usr/bin/bar
       when: foo_result.rc == 5
```

For more examples of using registered variables in conditions on later tasks, see *Conditionals*. Registered variables may be simple variables, list variables, dictionary variables, or complex nested data structures. The documentation for each module includes a `RETURN` section describing the return values for that module. To see the values for a particular task, run your playbook with `-v`.

Registered variables are stored in memory. You cannot cache registered variables for use in future playbook runs. Registered variables are only valid on the host for the rest of the current playbook run, including subsequent plays within the same playbook run.

Registered variables are host-level variables. When you register a variable in a task with a loop, the registered variable contains a value for each item in the loop. The data structure placed in the variable during the loop will contain a `results` attribute, that is a list of all responses from the module. For a more in-depth example of how this works, see the *Loops* section on using register with a loop.

Note: If a task fails or is skipped, Ansible still registers a variable with a failure or skipped status, unless the task is skipped based on tags. See *Tags* for information on adding and using tags.

Referencing nested variables

Many registered variables (and *facts*) are nested YAML or JSON data structures. You cannot access values from these nested data structures with the simple `{{ foo }}` syntax. You must use either bracket notation or dot notation. For example, to reference an IP address from your facts using the bracket notation:

```
{{ ansible_facts["eth0"]["ipv4"]["address"] }}
```

To reference an IP address from your facts using the dot notation:

```
{{ ansible_facts.eth0.ipv4.address }}
```

Transforming variables with Jinja2 filters

Jinja2 filters let you transform the value of a variable within a template expression. For example, the `capitalize` filter capitalizes any value passed to it; the `to_yaml` and `to_json` filters change the format of your variable values. Jinja2 includes many built-in filters and Ansible supplies many more filters. To find more examples of filters, see *Using filters to manipulate data*.

Where to set variables

You can define variables in a variety of places, such as in inventory, in playbooks, in reusable files, in roles, and at the command line. Ansible loads every possible variable it finds, then chooses the variable to apply based on *variable precedence rules*.

Defining variables in inventory

You can define different variables for each individual host, or set shared variables for a group of hosts in your inventory. For example, if all machines in the `[Boston]` group use 'boston.ntp.example.com' as an NTP server, you can set a group variable. The *How to build your inventory* page has details on setting *host variables* and *group variables* in inventory.

Defining variables in a play

You can define variables directly in a playbook play:

```yaml
- hosts: webservers
  vars:
    http_port: 80
```

When you define variables in a play, they are only visible to tasks executed in that play.

Defining variables in included files and roles

You can define variables in reusable variables files and/or in reusable roles. When you define variables in reusable variable files, the sensitive variables are separated from playbooks. This separation enables you to store your playbooks in a source control software and even share the playbooks, without the risk of exposing passwords or other sensitive and personal data. For information about creating reusable files and roles, see *Re-using Ansible artifacts*.

This example shows how you can include variables defined in an external file:

```yaml
---

- hosts: all
  remote_user: root
  vars:
    favcolor: blue
  vars_files:
    - /vars/external_vars.yml

  tasks:

```

(continues on next page)

(continued from previous page)

```
  - name: This is just a placeholder
    ansible.builtin.command: /bin/echo foo
```

The contents of each variables file is a simple YAML dictionary. For example:

```
---
# in the above example, this would be vars/external_vars.yml
somevar: somevalue
password: magic
```

Note: You can keep per-host and per-group variables in similar files. To learn about organizing your variables, see *Organizing host and group variables*.

Defining variables at runtime

You can define variables when you run your playbook by passing variables at the command line using the `--extra-vars` (or `-e`) argument. You can also request user input with a `vars_prompt` (see *Interactive input: prompts*). When you pass variables at the command line, use a single quoted string, that contains one or more variables, in one of the formats below.

key=value format

Values passed in using the `key=value` syntax are interpreted as strings. Use the JSON format if you need to pass non-string values such as Booleans, integers, floats, lists, and so on.

```
ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
```

JSON string format

```
ansible-playbook release.yml --extra-vars '{"version":"1.23.45","other_variable":"foo"}'
ansible-playbook arcade.yml --extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde
↪","sue"]}'
```

When passing variables with `--extra-vars`, you must escape quotes and other special characters appropriately for both your markup (for example, JSON), and for your shell:

```
ansible-playbook arcade.yml --extra-vars "{\"name\":\"Conan O\'Brien\"}"
ansible-playbook arcade.yml --extra-vars '{"name":"Conan O'\\\''Brien"}'
ansible-playbook script.yml --extra-vars "{\"dialog\":\"He said \\\"I just can\'t get␣
↪enough of those single and double-quotes"\!"\\\"\"}"
```

vars from a JSON or YAML file

If you have a lot of special characters, use a JSON or YAML file containing the variable definitions. Prepend both JSON and YAML filenames with @.

```
ansible-playbook release.yml --extra-vars "@some_file.json"
ansible-playbook release.yml --extra-vars "@some_file.yaml"
```

Variable precedence: Where should I put a variable?

You can set multiple variables with the same name in many different places. When you do this, Ansible loads every possible variable it finds, then chooses the variable to apply based on variable precedence. In other words, the different variables will override each other in a certain order.

Teams and projects that agree on guidelines for defining variables (where to define certain types of variables) usually avoid variable precedence concerns. We suggest that you define each variable in one place: figure out where to define a variable, and keep it simple. For examples, see *Tips on where to set variables*.

Some behavioral parameters that you can set in variables you can also set in Ansible configuration, as command-line options, and using playbook keywords. For example, you can define the user Ansible uses to connect to remote devices as a variable with `ansible_user`, in a configuration file with `DEFAULT_REMOTE_USER`, as a command-line option with `-u`, and with the playbook keyword `remote_user`. If you define the same parameter in a variable and by another method, the variable overrides the other setting. This approach allows host-specific settings to override more general settings. For examples and more details on the precedence of these various settings, see *Controlling how Ansible behaves: precedence rules*.

Understanding variable precedence

Ansible does apply variable precedence, and you might have a use for it. Here is the order of precedence from least to greatest (the last listed variables override all other variables):

1. command line values (for example, `-u my_user`, these are not variables)
2. role defaults (defined in role/defaults/main.yml)[1]
3. inventory file or script group vars[2]
4. inventory group_vars/all[3]
5. playbook group_vars/all[3]
6. inventory group_vars/*[3]
7. playbook group_vars/*[3]
8. inventory file or script host vars[2]
9. inventory host_vars/*[3]
10. playbook host_vars/*[3]
11. host facts / cached set_facts[4]
12. play vars

[1] Tasks in each role see their own role's defaults. Tasks defined outside of a role see the last role's defaults.
[2] Variables defined in inventory file or provided by dynamic inventory.
[3] Includes vars added by 'vars plugins' as well as host_vars and group_vars which are added by the default vars plugin shipped with Ansible.
[4] When created with set_facts's cacheable option, variables have the high precedence in the play, but are the same as a host facts precedence when they come from the cache.

13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (for example, `-e "user=my_user"`)(always win precedence)

In general, Ansible gives precedence to variables that were defined more recently, more actively, and with more explicit scope. Variables in the defaults folder inside a role are easily overridden. Anything in the vars directory of the role overrides previous versions of that variable in the namespace. Host and/or inventory variables override role defaults, but explicit includes such as the vars directory or an `include_vars` task override inventory variables.

Ansible merges different variables set in inventory so that more specific settings override more generic settings. For example, `ansible_ssh_user` specified as a group_var is overridden by `ansible_user` specified as a host_var. For details about the precedence of variables set in inventory, see *How variables are merged*.

Note: Within any section, redefining a var overrides the previous instance. If multiple groups have the same variable, the last one loaded wins. If you define a variable twice in a play's `vars:` section, the second one wins.

Note: The previous describes the default config `hash_behaviour=replace`, switch to `merge` to only partially overwrite.

Scoping variables

You can decide where to set a variable based on the scope you want that value to have. Ansible has three main scopes:

- Global: this is set by config, environment variables and the command line
- Play: each play and contained structures, vars entries (vars; vars_files; vars_prompt), role defaults and vars.
- Host: variables directly associated to a host, like inventory, include_vars, facts or registered task outputs

Inside a template, you automatically have access to all variables that are in scope for a host, plus any registered variables, facts, and magic variables.

Tips on where to set variables

You should choose where to define a variable based on the kind of control you might want over values.

Set variables in inventory that deal with geography or behavior. Since groups are frequently the entity that maps roles onto hosts, you can often set variables on the group instead of defining them on a role. Remember: child groups override parent groups, and host variables override group variables. See *Defining variables in inventory* for details on setting host and group variables.

Set common defaults in a `group_vars/all` file. See *Organizing host and group variables* for details on how to organize host and group variables in your inventory. Group variables are generally placed alongside your inventory file, but they can also be returned by dynamic inventory (see *Working with dynamic inventory*) or defined in AWX or on *Red Hat Ansible Automation Platform* from the UI or API:

```
---
# file: /etc/ansible/group_vars/all
# this is the site wide default
ntp_server: default-time.example.com
```

Set location-specific variables in `group_vars/my_location` files. All groups are children of the `all` group, so variables set here override those set in `group_vars/all`:

```
---
# file: /etc/ansible/group_vars/boston
ntp_server: boston-time.example.com
```

If one host used a different NTP server, you could set that in a host_vars file, which would override the group variable:

```
---
# file: /etc/ansible/host_vars/xyz.boston.example.com
ntp_server: override.example.com
```

Set defaults in roles to avoid undefined-variable errors. If you share your roles, other users can rely on the reasonable defaults you added in the `roles/x/defaults/main.yml` file, or they can easily override those values in inventory or at the command line. See *Roles* for more info. For example:

```
---
# file: roles/x/defaults/main.yml
# if no other value is supplied in inventory or as a parameter, this value will be used
http_port: 80
```

Set variables in roles to ensure a value is used in that role, and is not overridden by inventory variables. If you are not sharing your role with others, you can define app-specific behaviors like ports this way, in `roles/x/vars/main.yml`. If you are sharing roles with others, putting variables here makes them harder to override, although they still can by passing a parameter to the role or setting a variable with `-e`:

```
---
# file: roles/x/vars/main.yml
# this will absolutely be used in this role
http_port: 80
```

Pass variables as parameters when you call roles for maximum clarity, flexibility, and visibility. This approach overrides any defaults that exist for a role. For example:

```
roles:
  - role: apache
    vars:
       http_port: 8080
```

When you read this playbook it is clear that you have chosen to set a variable or override a default. You can also pass multiple values, which allows you to run the same role multiple times. See *Running a role multiple times in one play* for more details. For example:

```
roles:
  - role: app_user
    vars:
       myname: Ian
  - role: app_user
    vars:
      myname: Terry
  - role: app_user
    vars:
      myname: Graham
  - role: app_user
    vars:
      myname: John
```

Variables set in one role are available to later roles. You can set variables in a `roles/common_settings/vars/main.yml` file and use them in other roles and elsewhere in your playbook:

```
roles:
  - role: common_settings
  - role: something
    vars:
      foo: 12
  - role: something_else
```

Note: There are some protections in place to avoid the need to namespace variables. In this example, variables defined in 'common_settings' are available to 'something' and 'something_else' tasks, but tasks in 'something' have foo set at 12, even if 'common_settings' sets foo to 20.

Instead of worrying about variable precedence, we encourage you to think about how easily or how often you want to override a variable when deciding where to set it. If you are not sure what other variables are defined, and you need a particular value, use `--extra-vars` (`-e`) to override all other variables.

Using advanced variable syntax

For information about advanced YAML syntax used to declare variables and have more control over the data placed in YAML files used by Ansible, see *Advanced playbook syntax*.

See also:

Ansible playbooks
  An introduction to playbooks

Conditionals
  Conditional statements in playbooks

Using filters to manipulate data
    Jinja2 filters and their uses

Loops
    Looping in playbooks

Roles
    Playbook organization by roles

General tips
    Tips and tricks for playbooks

Special Variables
    List of special variables

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Discovering variables: facts and magic variables

With Ansible you can retrieve or discover certain variables containing information about your remote systems or about Ansible itself. Variables related to remote systems are called facts. With facts, you can use the behavior or state of one system as configuration on other systems. For example, you can use the IP address of one system as a configuration value on another system. Variables related to Ansible are called magic variables.

Ansible facts

Ansible facts are data related to your remote systems, including operating systems, IP addresses, attached filesystems, and more. You can access this data in the `ansible_facts` variable. By default, you can also access some Ansible facts as top-level variables with the `ansible_` prefix. You can disable this behavior using the *INJECT_FACTS_AS_VARS* setting. To see all available facts, add this task to a play:

```yaml
- name: Print all available facts
  ansible.builtin.debug:
    var: ansible_facts
```

To see the 'raw' information as gathered, run this command at the command line:

```
ansible <hostname> -m ansible.builtin.setup
```

Facts include a large amount of variable data, which may look like this:

```
{
    "ansible_all_ipv4_addresses": [
        "REDACTED IP ADDRESS"
    ],
    "ansible_all_ipv6_addresses": [
        "REDACTED IPV6 ADDRESS"
    ],
    "ansible_apparmor": {
        "status": "disabled"
    },
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "11/28/2013",
    "ansible_bios_version": "4.1.5",
    "ansible_cmdline": {
        "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-862.14.4.el7.x86_64",
        "console": "ttyS0,115200",
        "no_timer_check": true,
        "nofb": true,
        "nomodeset": true,
        "ro": true,
        "root": "LABEL=cloudimg-rootfs",
        "vga": "normal"
    },
    "ansible_date_time": {
        "date": "2018-10-25",
        "day": "25",
        "epoch": "1540469324",
        "hour": "12",
        "iso8601": "2018-10-25T12:08:44Z",
        "iso8601_basic": "20181025T120844109754",
        "iso8601_basic_short": "20181025T120844",
        "iso8601_micro": "2018-10-25T12:08:44.109968Z",
        "minute": "08",
        "month": "10",
        "second": "44",
        "time": "12:08:44",
        "tz": "UTC",
        "tz_offset": "+0000",
        "weekday": "Thursday",
        "weekday_number": "4",
        "weeknumber": "43",
        "year": "2018"
    },
    "ansible_default_ipv4": {
        "address": "REDACTED",
        "alias": "eth0",
        "broadcast": "REDACTED",
        "gateway": "REDACTED",
        "interface": "eth0",
```

(continues on next page)

(continued from previous page)

```
        "macaddress": "REDACTED",
        "mtu": 1500,
        "netmask": "255.255.255.0",
        "network": "REDACTED",
        "type": "ether"
    },
    "ansible_default_ipv6": {},
    "ansible_device_links": {
        "ids": {},
        "labels": {
            "xvda1": [
                "cloudimg-rootfs"
            ],
            "xvdd": [
                "config-2"
            ]
        },
        "masters": {},
        "uuids": {
            "xvda1": [
                "cac81d61-d0f8-4b47-84aa-b48798239164"
            ],
            "xvdd": [
                "2018-10-25-12-05-57-00"
            ]
        }
    },
    "ansible_devices": {
        "xvda": {
            "holders": [],
            "host": "",
            "links": {
                "ids": [],
                "labels": [],
                "masters": [],
                "uuids": []
            },
            "model": null,
            "partitions": {
                "xvda1": {
                    "holders": [],
                    "links": {
                        "ids": [],
                        "labels": [
                            "cloudimg-rootfs"
                        ],
                        "masters": [],
                        "uuids": [
                            "cac81d61-d0f8-4b47-84aa-b48798239164"
                        ]
                    },
                    "sectors": "83883999",
```

(continues on next page)

(continued from previous page)

```
                    "sectorsize": 512,
                    "size": "40.00 GB",
                    "start": "2048",
                    "uuid": "cac81d61-d0f8-4b47-84aa-b48798239164"
                }
            },
            "removable": "0",
            "rotational": "0",
            "sas_address": null,
            "sas_device_handle": null,
            "scheduler_mode": "deadline",
            "sectors": "83886080",
            "sectorsize": "512",
            "size": "40.00 GB",
            "support_discard": "0",
            "vendor": null,
            "virtual": 1
        },
        "xvdd": {
            "holders": [],
            "host": "",
            "links": {
                "ids": [],
                "labels": [
                    "config-2"
                ],
                "masters": [],
                "uuids": [
                    "2018-10-25-12-05-57-00"
                ]
            },
            "model": null,
            "partitions": {},
            "removable": "0",
            "rotational": "0",
            "sas_address": null,
            "sas_device_handle": null,
            "scheduler_mode": "deadline",
            "sectors": "131072",
            "sectorsize": "512",
            "size": "64.00 MB",
            "support_discard": "0",
            "vendor": null,
            "virtual": 1
        },
        "xvde": {
            "holders": [],
            "host": "",
            "links": {
                "ids": [],
                "labels": [],
                "masters": [],
```

(continues on next page)

(continued from previous page)

```
                "uuids": []
            },
            "model": null,
            "partitions": {
                "xvde1": {
                    "holders": [],
                    "links": {
                        "ids": [],
                        "labels": [],
                        "masters": [],
                        "uuids": []
                    },
                    "sectors": "167770112",
                    "sectorsize": 512,
                    "size": "80.00 GB",
                    "start": "2048",
                    "uuid": null
                }
            },
            "removable": "0",
            "rotational": "0",
            "sas_address": null,
            "sas_device_handle": null,
            "scheduler_mode": "deadline",
            "sectors": "167772160",
            "sectorsize": "512",
            "size": "80.00 GB",
            "support_discard": "0",
            "vendor": null,
            "virtual": 1
        }
    },
    "ansible_distribution": "CentOS",
    "ansible_distribution_file_parsed": true,
    "ansible_distribution_file_path": "/etc/redhat-release",
    "ansible_distribution_file_variety": "RedHat",
    "ansible_distribution_major_version": "7",
    "ansible_distribution_release": "Core",
    "ansible_distribution_version": "7.5.1804",
    "ansible_dns": {
        "nameservers": [
            "127.0.0.1"
        ]
    },
    "ansible_domain": "",
    "ansible_effective_group_id": 1000,
    "ansible_effective_user_id": 1000,
    "ansible_env": {
        "HOME": "/home/zuul",
        "LANG": "en_US.UTF-8",
        "LESSOPEN": "||/usr/bin/lesspipe.sh %s",
        "LOGNAME": "zuul",
```

(continues on next page)

(continued from previous page)

```
        "MAIL": "/var/mail/zuul",
        "PATH": "/usr/local/bin:/usr/bin",
        "PWD": "/home/zuul",
        "SELINUX_LEVEL_REQUESTED": "",
        "SELINUX_ROLE_REQUESTED": "",
        "SELINUX_USE_CURRENT_RANGE": "",
        "SHELL": "/bin/bash",
        "SHLVL": "2",
        "SSH_CLIENT": "REDACTED 55672 22",
        "SSH_CONNECTION": "REDACTED 55672 REDACTED 22",
        "USER": "zuul",
        "XDG_RUNTIME_DIR": "/run/user/1000",
        "XDG_SESSION_ID": "1",
        "_": "/usr/bin/python2"
    },
    "ansible_eth0": {
        "active": true,
        "device": "eth0",
        "ipv4": {
            "address": "REDACTED",
            "broadcast": "REDACTED",
            "netmask": "255.255.255.0",
            "network": "REDACTED"
        },
        "ipv6": [
            {
                "address": "REDACTED",
                "prefix": "64",
                "scope": "link"
            }
        ],
        "macaddress": "REDACTED",
        "module": "xen_netfront",
        "mtu": 1500,
        "pciid": "vif-0",
        "promisc": false,
        "type": "ether"
    },
    "ansible_eth1": {
        "active": true,
        "device": "eth1",
        "ipv4": {
            "address": "REDACTED",
            "broadcast": "REDACTED",
            "netmask": "255.255.224.0",
            "network": "REDACTED"
        },
        "ipv6": [
            {
                "address": "REDACTED",
                "prefix": "64",
                "scope": "link"
```

(continues on next page)

(continued from previous page)

```
            }
        ],
        "macaddress": "REDACTED",
        "module": "xen_netfront",
        "mtu": 1500,
        "pciid": "vif-1",
        "promisc": false,
        "type": "ether"
    },
    "ansible_fips": false,
    "ansible_form_factor": "Other",
    "ansible_fqdn": "centos-7-rax-dfw-0003427354",
    "ansible_hostname": "centos-7-rax-dfw-0003427354",
    "ansible_interfaces": [
        "lo",
        "eth1",
        "eth0"
    ],
    "ansible_is_chroot": false,
    "ansible_kernel": "3.10.0-862.14.4.el7.x86_64",
    "ansible_lo": {
        "active": true,
        "device": "lo",
        "ipv4": {
            "address": "127.0.0.1",
            "broadcast": "host",
            "netmask": "255.0.0.0",
            "network": "127.0.0.0"
        },
        "ipv6": [
            {
                "address": "::1",
                "prefix": "128",
                "scope": "host"
            }
        ],
        "mtu": 65536,
        "promisc": false,
        "type": "loopback"
    },
    "ansible_local": {},
    "ansible_lsb": {
        "codename": "Core",
        "description": "CentOS Linux release 7.5.1804 (Core)",
        "id": "CentOS",
        "major_release": "7",
        "release": "7.5.1804"
    },
    "ansible_machine": "x86_64",
    "ansible_machine_id": "2db133253c984c82aef2fafcce6f2bed",
    "ansible_memfree_mb": 7709,
    "ansible_memory_mb": {
```

(continues on next page)

(continued from previous page)

```
        "nocache": {
            "free": 7804,
            "used": 173
        },
        "real": {
            "free": 7709,
            "total": 7977,
            "used": 268
        },
        "swap": {
            "cached": 0,
            "free": 0,
            "total": 0,
            "used": 0
        }
    },
    "ansible_memtotal_mb": 7977,
    "ansible_mounts": [
        {
            "block_available": 7220998,
            "block_size": 4096,
            "block_total": 9817227,
            "block_used": 2596229,
            "device": "/dev/xvda1",
            "fstype": "ext4",
            "inode_available": 10052341,
            "inode_total": 10419200,
            "inode_used": 366859,
            "mount": "/",
            "options": "rw,seclabel,relatime,data=ordered",
            "size_available": 29577207808,
            "size_total": 40211361792,
            "uuid": "cac81d61-d0f8-4b47-84aa-b48798239164"
        },
        {
            "block_available": 0,
            "block_size": 2048,
            "block_total": 252,
            "block_used": 252,
            "device": "/dev/xvdd",
            "fstype": "iso9660",
            "inode_available": 0,
            "inode_total": 0,
            "inode_used": 0,
            "mount": "/mnt/config",
            "options": "ro,relatime,mode=0700",
            "size_available": 0,
            "size_total": 516096,
            "uuid": "2018-10-25-12-05-57-00"
        }
    ],
    "ansible_nodename": "centos-7-rax-dfw-0003427354",
```

(continues on next page)

(continued from previous page)

```
    "ansible_os_family": "RedHat",
    "ansible_pkg_mgr": "yum",
    "ansible_processor": [
        "0",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "1",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "2",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "3",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "4",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "5",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "6",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz",
        "7",
        "GenuineIntel",
        "Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz"
    ],
    "ansible_processor_cores": 8,
    "ansible_processor_count": 8,
    "ansible_processor_nproc": 8,
    "ansible_processor_threads_per_core": 1,
    "ansible_processor_vcpus": 8,
    "ansible_product_name": "HVM domU",
    "ansible_product_serial": "REDACTED",
    "ansible_product_uuid": "REDACTED",
    "ansible_product_version": "4.1.5",
    "ansible_python": {
        "executable": "/usr/bin/python2",
        "has_sslcontext": true,
        "type": "CPython",
        "version": {
            "major": 2,
            "micro": 5,
            "minor": 7,
            "releaselevel": "final",
            "serial": 0
        },
        "version_info": [
            2,
            7,
            5,
```

(continues on next page)

(continued from previous page)

```
            "final",
            0
        ]
    },
    "ansible_python_version": "2.7.5",
    "ansible_real_group_id": 1000,
    "ansible_real_user_id": 1000,
    "ansible_selinux": {
        "config_mode": "enforcing",
        "mode": "enforcing",
        "policyvers": 31,
        "status": "enabled",
        "type": "targeted"
    },
    "ansible_selinux_python_present": true,
    "ansible_service_mgr": "systemd",
    "ansible_ssh_host_key_ecdsa_public": "REDACTED KEY VALUE",
    "ansible_ssh_host_key_ed25519_public": "REDACTED KEY VALUE",
    "ansible_ssh_host_key_rsa_public": "REDACTED KEY VALUE",
    "ansible_swapfree_mb": 0,
    "ansible_swaptotal_mb": 0,
    "ansible_system": "Linux",
    "ansible_system_capabilities": [
        ""
    ],
    "ansible_system_capabilities_enforced": "True",
    "ansible_system_vendor": "Xen",
    "ansible_uptime_seconds": 151,
    "ansible_user_dir": "/home/zuul",
    "ansible_user_gecos": "",
    "ansible_user_gid": 1000,
    "ansible_user_id": "zuul",
    "ansible_user_shell": "/bin/bash",
    "ansible_user_uid": 1000,
    "ansible_userspace_architecture": "x86_64",
    "ansible_userspace_bits": "64",
    "ansible_virtualization_role": "guest",
    "ansible_virtualization_type": "xen",
    "gather_subset": [
        "all"
    ],
    "module_setup": true
}
```

You can reference the model of the first disk in the facts shown above in a template or playbook as:

```
{{ ansible_facts['devices']['xvda']['model'] }}
```

To reference the system hostname:

```
{{ ansible_facts['nodename'] }}
```

You can use facts in conditionals (see *Conditionals*) and also in templates. You can also use facts to create dynamic groups of hosts that match particular criteria, see the group_by module documentation for details.

Note: Because `ansible_date_time` is created and cached when Ansible gathers facts before each playbook run, it can get stale with long-running playbooks. If your playbook takes a long time to run, use the `pipe` filter (for example, `lookup('pipe', 'date +%Y-%m-%d.%H:%M:%S')`) or *now()* with a Jinja 2 template instead of `ansible_date_time`.

Package requirements for fact gathering

On some distros, you may see missing fact values or facts set to default values because the packages that support gathering those facts are not installed by default. You can install the necessary packages on your remote hosts using the OS package manager. Known dependencies include:

- Linux Network fact gathering - Depends on the `ip` binary, commonly included in the `iproute2` package.

Caching facts

Like registered variables, facts are stored in memory by default. However, unlike registered variables, facts can be gathered independently and cached for repeated use. With cached facts, you can refer to facts from one system when configuring a second system, even if Ansible executes the current play on the second system first. For example:

```
{{ hostvars['asdf.example.com']['ansible_facts']['os_family'] }}
```

Caching is controlled by the cache plugins. By default, Ansible uses the memory cache plugin, which stores facts in memory for the duration of the current playbook run. To retain Ansible facts for repeated use, select a different cache plugin. See *Cache plugins* for details.

Fact caching can improve performance. If you manage thousands of hosts, you can configure fact caching to run nightly, then manage configuration on a smaller set of servers periodically throughout the day. With cached facts, you have access to variables and information about all hosts even when you are only managing a small number of servers.

Disabling facts

By default, Ansible gathers facts at the beginning of each play. If you do not need to gather facts (for example, if you know everything about your systems centrally), you can turn off fact gathering at the play level to improve scalability. Disabling facts may particularly improve performance in push mode with very large numbers of systems, or if you are using Ansible on experimental platforms. To disable fact gathering:

```
- hosts: whatever
  gather_facts: false
```

Adding custom facts

The setup module in Ansible automatically discovers a standard set of facts about each host. If you want to add custom values to your facts, you can write a custom facts module, set temporary facts with a `ansible.builtin.set_fact` task, or provide permanent custom facts using the facts.d directory.

facts.d or local facts

New in version 1.3.

You can add static custom facts by adding static files to facts.d, or add dynamic facts by adding executable scripts to facts.d. For example, you can add a list of all users on a host to your facts by creating and running a script in facts.d.

To use facts.d, create an `/etc/ansible/facts.d` directory on the remote host or hosts. If you prefer a different directory, create it and specify it using the `fact_path` play keyword. Add files to the directory to supply your custom facts. All file names must end with `.fact`. The files can be JSON, INI, or executable files returning JSON.

To add static facts, simply add a file with the `.fact` extension. For example, create `/etc/ansible/facts.d/preferences.fact` with this content:

```
[general]
asdf=1
bar=2
```

Note: Make sure the file is not executable as this will break the `ansible.builtin.setup` module.

The next time fact gathering runs, your facts will include a hash variable fact named `general` with `asdf` and `bar` as members. To validate this, run the following:

```
ansible <hostname> -m ansible.builtin.setup -a "filter=ansible_local"
```

And you will see your custom fact added:

```
{
    "ansible_local": {
        "preferences": {
            "general": {
                "asdf" : "1",
                "bar"  : "2"
            }
        }
    }
}
```

The ansible_local namespace separates custom facts created by facts.d from system facts or variables defined elsewhere in the playbook, so variables will not override each other. You can access this custom fact in a template or playbook as:

```
{{ ansible_local['preferences']['general']['asdf'] }}
```

Note: The key part in the key=value pairs will be converted into lowercase inside the ansible_local variable. Using the example above, if the ini file contained `XYZ=3` in the `[general]` section, then you should expect to access it as: `{{ ansible_local['preferences']['general']['xyz'] }}` and not `{{ ansible_local['preferences']['general']['XYZ'] }}`. This is because Ansible uses Python's ConfigParser which passes all option names through the optionxform method and this method's default implementation converts option names to lower case.

You can also use facts.d to execute a script on the remote host, generating dynamic custom facts to the ansible_local namespace. For example, you can generate a list of all users that exist on a remote host as a fact about that host. To generate dynamic custom facts using facts.d:

1. Write and test a script to generate the JSON data you want.
2. Save the script in your facts.d directory.
3. Make sure your script has the `.fact` file extension.
4. Make sure your script is executable by the Ansible connection user.
5. Gather facts to execute the script and add the JSON output to ansible_local.

By default, fact gathering runs once at the beginning of each play. If you create a custom fact using facts.d in a playbook, it will be available in the next play that gathers facts. If you want to use it in the same play where you created it, you must explicitly re-run the setup module. For example:

```
- hosts: webservers
  tasks:

    - name: Create directory for ansible custom facts
      ansible.builtin.file:
        state: directory
        recurse: true
        path: /etc/ansible/facts.d

    - name: Install custom ipmi fact
      ansible.builtin.copy:
        src: ipmi.fact
        dest: /etc/ansible/facts.d

    - name: Re-read facts after adding custom fact
      ansible.builtin.setup:
        filter: ansible_local
```

If you use this pattern frequently, a custom facts module would be more efficient than facts.d.

Information about Ansible: magic variables

You can access information about Ansible operations, including the python version being used, the hosts and groups in inventory, and the directories for playbooks and roles, using "magic" variables. Like connection variables, magic variables are *Special Variables*. Magic variable names are reserved - do not set variables with these names. The variable `environment` is also reserved.

The most commonly used magic variables are `hostvars`, `groups`, `group_names`, and `inventory_hostname`. With `hostvars`, you can access variables defined for any host in the play, at any point in a playbook. You can access Ansible facts using the `hostvars` variable too, but only after you have gathered (or cached) facts. Note that variables defined at play objects are not defined for specific hosts and therefore are not mapped to hostvars.

If you want to configure your database server using the value of a 'fact' from another node, or the value of an inventory variable assigned to another node, you can use `hostvars` in a template or on an action line:

```
{{ hostvars['test.example.com']['ansible_facts']['distribution'] }}
```

With groups, a list of all the groups (and hosts) in the inventory, you can enumerate all hosts within a group. For example:

```
{% for host in groups['app_servers'] %}
   # something that applies to all app servers.
{% endfor %}
```

You can use groups and hostvars together to find all the IP addresses in a group.

```
{% for host in groups['app_servers'] %}
   {{ hostvars[host]['ansible_facts']['eth0']['ipv4']['address'] }}
{% endfor %}
```

You can use this approach to point a frontend proxy server to all the hosts in your app servers group, to set up the correct firewall rules between servers, and so on. You must either cache facts or gather facts for those hosts before the task that fills out the template.

With group_names, a list (array) of all the groups the current host is in, you can create templated files that vary based on the group membership (or role) of the host:

```
{% if 'webserver' in group_names %}
   # some part of a configuration file that only applies to webservers
{% endif %}
```

You can use the magic variable inventory_hostname, the name of the host as configured in your inventory, as an alternative to ansible_hostname when fact-gathering is disabled. If you have a long FQDN, you can use inventory_hostname_short, which contains the part up to the first period, without the rest of the domain.

Other useful magic variables refer to the current play or playbook. These vars may be useful for filling out templates with multiple hostnames or for injecting the list into the rules for a load balancer.

ansible_play_hosts is the list of all hosts still active in the current play.

ansible_play_batch is a list of hostnames that are in scope for the current 'batch' of the play.

The batch size is defined by serial, when not set it is equivalent to the whole play (making it the same as ansible_play_hosts).

ansible_playbook_python is the path to the python executable used to invoke the Ansible command line tool.

inventory_dir is the pathname of the directory holding Ansible's inventory host file.

inventory_file is the pathname and the filename pointing to the Ansible's inventory host file.

playbook_dir contains the playbook base directory.

role_path contains the current role's pathname and only works inside a role.

ansible_check_mode is a boolean, set to True if you run Ansible with --check.

Ansible version

New in version 1.8.

To adapt playbook behavior to different versions of Ansible, you can use the variable `ansible_version`, which has the following structure:

```
{
    "ansible_version": {
        "full": "2.10.1",
        "major": 2,
        "minor": 10,
        "revision": 1,
        "string": "2.10.1"
    }
}
```

Playbook Example: Continuous Delivery and Rolling Upgrades

What is continuous delivery?

Continuous delivery (CD) means frequently delivering updates to your software application.

The idea is that by updating more often, you do not have to wait for a specific timed period, and your organization gets better at the process of responding to change.

Some Ansible users are deploying updates to their end users on an hourly or even more frequent basis – sometimes every time there is an approved code change. To achieve this, you need tools to be able to quickly apply those updates in a zero-downtime way.

This document describes in detail how to achieve this goal, using one of Ansible's most complete example playbooks as a template: lamp_haproxy. This example uses a lot of Ansible features: roles, templates, and group variables, and it also comes with an orchestration playbook that can do zero-downtime rolling upgrades of the web application stack.

Note: Click here for the latest playbooks for this example.

The playbooks deploy Apache, PHP, MySQL, Nagios, and HAProxy to a CentOS-based set of servers.

We're not going to cover how to run these playbooks here. Read the included README in the github project along with the example for that information. Instead, we're going to take a close look at every part of the playbook and describe what it does.

Site deployment

Let's start with `site.yml`. This is our site-wide deployment playbook. It can be used to initially deploy the site, as well as push updates to all of the servers:

```
---
# This playbook deploys the whole application stack in this site.

# Apply common configuration to all hosts
- hosts: all

  roles:
  - common

# Configure and deploy database servers.
- hosts: dbservers

  roles:
  - db

# Configure and deploy the web servers. Note that we include two roles
# here, the 'base-apache' role which simply sets up Apache, and 'web'
# which includes our example web application.

- hosts: webservers

  roles:
  - base-apache
  - web

# Configure and deploy the load balancer(s).
- hosts: lbservers

  roles:
  - haproxy

# Configure and deploy the Nagios monitoring node(s).
- hosts: monitoring

  roles:
  - base-apache
  - nagios
```

Note: If you're not familiar with terms like playbooks and plays, you should review *Working with playbooks*.

In this playbook we have 5 plays. The first one targets `all` hosts and applies the `common` role to all of the hosts. This is for site-wide things like yum repository configuration, firewall configuration, and anything else that needs to apply to all of the servers.

The next four plays run against specific host groups and apply specific roles to those servers. Along with the roles for Nagios monitoring, the database, and the web application, we've implemented a `base-apache` role that installs and configures a basic Apache setup. This is used by both the sample web application and the Nagios hosts.

Reusable content: roles

By now you should have a bit of understanding about roles and how they work in Ansible. Roles are a way to organize content: tasks, handlers, templates, and files, into reusable components.

This example has six roles: `common`, `base-apache`, `db`, `haproxy`, `nagios`, and `web`. How you organize your roles is up to you and your application, but most sites will have one or more common roles that are applied to all systems, and then a series of application-specific roles that install and configure particular parts of the site.

Roles can have variables and dependencies, and you can pass in parameters to roles to modify their behavior. You can read more about roles in the *Roles* section.

Configuration: group variables

Group variables are variables that are applied to groups of servers. They can be used in templates and in playbooks to customize behavior and to provide easily-changed settings and parameters. They are stored in a directory called `group_vars` in the same location as your inventory. Here is lamp_haproxy's `group_vars/all` file. As you might expect, these variables are applied to all of the machines in your inventory:

```
---
httpd_port: 80
ntpserver: 192.0.2.23
```

This is a YAML file, and you can create lists and dictionaries for more complex variable structures. In this case, we are just setting two variables, one for the port for the web server, and one for the NTP server that our machines should use for time synchronization.

Here's another group variables file. This is `group_vars/dbservers` which applies to the hosts in the `dbservers` group:

```
---
mysqlservice: mysqld
mysql_port: 3306
dbuser: root
dbname: foodb
upassword: usersecret
```

If you look in the example, there are group variables for the `webservers` group and the `lbservers` group, similarly.

These variables are used in a variety of places. You can use them in playbooks, like this, in `roles/db/tasks/main.yml`:

```
- name: Create Application Database
  mysql_db:
    name: "{{ dbname }}"
    state: present

- name: Create Application DB User
  mysql_user:
    name: "{{ dbuser }}"
```

(continues on next page)

(continued from previous page)

```
    password: "{{ upassword }}"
    priv: "*.*:ALL"
    host: '%'
    state: present
```

You can also use these variables in templates, like this, in `roles/common/templates/ntp.conf.j2`:

```
driftfile /var/lib/ntp/drift

restrict 127.0.0.1
restrict -6 ::1

server {{ ntpserver }}

includefile /etc/ntp/crypto/pw

keys /etc/ntp/keys
```

You can see that the variable substitution syntax of {{ and }} is the same for both templates and variables. The syntax inside the curly braces is Jinja2, and you can do all sorts of operations and apply different filters to the data inside. In templates, you can also use for loops and if statements to handle more complex situations, like this, in `roles/common/templates/iptables.j2`:

```
{% if inventory_hostname in groups['dbservers'] %}
-A INPUT -p tcp  --dport 3306 -j  ACCEPT
{% endif %}
```

This is testing to see if the inventory name of the machine we're currently operating on (`inventory_hostname`) exists in the inventory group `dbservers`. If so, that machine will get an iptables ACCEPT line for port 3306.

Here's another example, from the same template:

```
{% for host in groups['monitoring'] %}
-A INPUT -p tcp -s {{ hostvars[host].ansible_default_ipv4.address }} --dport 5666 -j␣
↪ACCEPT
{% endfor %}
```

This loops over all of the hosts in the group called `monitoring`, and adds an ACCEPT line for each monitoring hosts' default IPv4 address to the current machine's iptables configuration, so that Nagios can monitor those hosts.

You can learn a lot more about Jinja2 and its capabilities here, and you can read more about Ansible variables in general in the *Using Variables* section.

The rolling upgrade

Now you have a fully-deployed site with web servers, a load balancer, and monitoring. How do you update it? This is where Ansible's orchestration features come into play. While some applications use the term 'orchestration' to mean basic ordering or command-blasting, Ansible refers to orchestration as 'conducting machines like an orchestra', and has a pretty sophisticated engine for it.

Ansible has the capability to do operations on multi-tier applications in a coordinated way, making it easy to orchestrate a sophisticated zero-downtime rolling upgrade of our web application. This is implemented in a separate playbook, called `rolling_update.yml`.

Looking at the playbook, you can see it is made up of two plays. The first play is very simple and looks like this:

```
- hosts: monitoring
  tasks: []
```

What's going on here, and why are there no tasks? You might know that Ansible gathers "facts" from the servers before operating upon them. These facts are useful for all sorts of things: networking information, OS/distribution versions, and so on. In our case, we need to know something about all of the monitoring servers in our environment before we perform the update, so this simple play forces a fact-gathering step on our monitoring servers. You will see this pattern sometimes, and it's a useful trick to know.

The next part is the update play. The first part looks like this:

```
- hosts: webservers
  user: root
  serial: 1
```

This is just a normal play definition, operating on the `webservers` group. The `serial` keyword tells Ansible how many servers to operate on at once. If it's not specified, Ansible will parallelize these operations up to the default "forks" limit specified in the configuration file. But for a zero-downtime rolling upgrade, you may not want to operate on that many hosts at once. If you had just a handful of webservers, you may want to set `serial` to 1, for one host at a time. If you have 100, maybe you could set `serial` to 10, for ten at a time.

Here is the next part of the update play:

```
pre_tasks:
- name: disable nagios alerts for this host webserver service
  nagios:
    action: disable_alerts
    host: "{{ inventory_hostname }}"
    services: webserver
  delegate_to: "{{ item }}"
  loop: "{{ groups.monitoring }}"

- name: disable the server in haproxy
  shell: echo "disable server myapplb/{{ inventory_hostname }}" | socat stdio /var/lib/
↪haproxy/stats
  delegate_to: "{{ item }}"
  loop: "{{ groups.lbservers }}"
```

Note:

- The `serial` keyword forces the play to be executed in 'batches'. Each batch counts as a full play with a subselection of hosts. This has some consequences on play behavior. For example, if all hosts in a batch fails, the play fails, which in turn fails the entire run. You should consider this when combining with `max_fail_percentage`.

The `pre_tasks` keyword just lets you list tasks to run before the roles are called. This will make more sense in a minute. If you look at the names of these tasks, you can see that we are disabling Nagios alerts and then removing the webserver that we are currently updating from the HAProxy load balancing pool.

The `delegate_to` and `loop` arguments, used together, cause Ansible to loop over each monitoring server and load balancer, and perform that operation (delegate that operation) on the monitoring or load balancing server, "on behalf" of the webserver. In programming terms, the outer loop is the list of web servers, and the inner loop is the list of monitoring servers.

Note that the HAProxy step looks a little complicated. We're using HAProxy in this example because it's freely available, though if you have (for instance) an F5 or Netscaler in your infrastructure (or maybe you have an AWS Elastic IP setup?), you can use Ansible modules to communicate with them instead. You might also wish to use other monitoring modules instead of nagios, but this just shows the main goal of the 'pre tasks' section – take the server out of monitoring, and take it out of rotation.

The next step simply re-applies the proper roles to the web servers. This will cause any configuration management declarations in `web` and `base-apache` roles to be applied to the web servers, including an update of the web application code itself. We don't have to do it this way–we could instead just purely update the web application, but this is a good example of how roles can be used to reuse tasks:

```
roles:
- common
- base-apache
- web
```

Finally, in the `post_tasks` section, we reverse the changes to the Nagios configuration and put the web server back in the load balancing pool:

```
post_tasks:
- name: Enable the server in haproxy
  shell: echo "enable server myapplb/{{ inventory_hostname }}" | socat stdio /var/lib/
↪haproxy/stats
  delegate_to: "{{ item }}"
  loop: "{{ groups.lbservers }}"

- name: re-enable nagios alerts
  nagios:
    action: enable_alerts
    host: "{{ inventory_hostname }}"
    services: webserver
  delegate_to: "{{ item }}"
  loop: "{{ groups.monitoring }}"
```

Again, if you were using a Netscaler or F5 or Elastic Load Balancer, you would just substitute in the appropriate modules instead.

Managing other load balancers

In this example, we use the simple HAProxy load balancer to front-end the web servers. It's easy to configure and easy to manage. As we have mentioned, Ansible has support for a variety of other load balancers like Citrix NetScaler, F5 BigIP, Amazon Elastic Load Balancers, and more. See the Working With Modules documentation for more information.

For other load balancers, you may need to send shell commands to them (like we do for HAProxy above), or call an API, if your load balancer exposes one. For the load balancers for which Ansible has modules, you may want to run them as a `local_action` if they contact an API. You can read more about local actions in the *Controlling where tasks run: delegation and local actions* section. Should you develop anything interesting for some hardware where there is not a module, it might make for a good contribution!

Continuous delivery end-to-end

Now that you have an automated way to deploy updates to your application, how do you tie it all together? A lot of organizations use a continuous integration tool like Jenkins or Atlassian Bamboo to tie the development, test, release, and deploy steps together. You may also want to use a tool like Gerrit to add a code review step to commits to either the application code itself, or to your Ansible playbooks, or both.

Depending on your environment, you might be deploying continuously to a test environment, running an integration test battery against that environment, and then deploying automatically into production. Or you could keep it simple and just use the rolling-update for on-demand deployment into test or production specifically. This is all up to you.

For integration with Continuous Integration systems, you can easily trigger playbook runs using the `ansible-playbook` command line tool, or, if you're using AWX, the `tower-cli` command or the built-in REST API. (The tower-cli command 'joblaunch' will spawn a remote job over the REST API and is pretty slick).

This should give you a good idea of how to structure a multi-tier application with Ansible, and orchestrate operations upon that app, with the eventual goal of continuous delivery to your customers. You could extend the idea of the rolling upgrade to lots of different parts of the app; maybe add front-end web servers along with application servers, for instance, or replace the SQL database with something like MongoDB or Riak. Ansible gives you the capability to easily manage complicated environments and automate common operations.

See also:

lamp_haproxy example
    The lamp_haproxy example discussed here.

Working with playbooks
    An introduction to playbooks

Roles
    An introduction to playbook roles

Using Variables
    An introduction to Ansible variables

Ansible.com: Continuous Delivery
    An introduction to Continuous Delivery with Ansible

1.6.3 Executing playbooks

Ready to run your Ansible playbook?

Running complex playbooks requires some trial and error so learn about some of the abilities that Ansible gives you to ensure successful execution. You can validate your tasks with "dry run" playbooks, use the start-at-task and step mode options to efficiently troubleshoot playbooks. You can also use Ansible debugger to correct tasks during execution. Ansible also offers flexibility with asynchronous playbook execution and tags that let you run specific parts of your playbook.

Validating tasks: check mode and diff mode

Ansible provides two modes of execution that validate tasks: check mode and diff mode. These modes can be used separately or together. They are useful when you are creating or editing a playbook or role and you want to know what it will do. In check mode, Ansible runs without making any changes on remote systems. Modules that support check mode report the changes they would have made. Modules that do not support check mode report nothing and do nothing. In diff mode, Ansible provides before-and-after comparisons. Modules that support diff mode display detailed information. You can combine check mode and diff mode for detailed validation of your playbook or role.

Using check mode

Check mode is just a simulation. It will not generate output for tasks that use *conditionals based on registered variables* (results of prior tasks). However, it is great for validating configuration management playbooks that run on one node at a time. To run a playbook in check mode:

```
ansible-playbook foo.yml --check
```

Enforcing or preventing check mode on tasks

New in version 2.2.

If you want certain tasks to run in check mode always, or never, regardless of whether you run the playbook with or without `--check`, you can add the `check_mode` option to those tasks:

- To force a task to run in check mode, even when the playbook is called without `--check`, set `check_mode: true`.
- To force a task to run in normal mode and make changes to the system, even when the playbook is called with `--check`, set `check_mode:  false`.

For example:

```
tasks:
  - name: This task will always make changes to the system
    ansible.builtin.command: /something/to/run --even-in-check-mode
    check_mode: false

  - name: This task will never make changes to the system
    ansible.builtin.lineinfile:
      line: "important config"
      dest: /path/to/myconfig.conf
      state: present
    check_mode: true
    register: changes_to_important_config
```

Running single tasks with `check_mode: true` can be useful for testing Ansible modules, either to test the module itself or to test the conditions under which a module would make changes. You can register variables (see *Conditionals*) on these tasks for even more detail on the potential changes.

Note: Prior to version 2.2 only the equivalent of `check_mode: false` existed. The notation for that was `always_run: yes`.

Skipping tasks or ignoring errors in check mode

New in version 2.1.

If you want to skip a task or ignore errors on a task when you run Ansible in check mode, you can use a boolean magic variable `ansible_check_mode`, which is set to `True` when Ansible runs in check mode. For example:

```yaml
tasks:

  - name: This task will be skipped in check mode
    ansible.builtin.git:
      repo: ssh://git@github.com/mylogin/hello.git
      dest: /home/mylogin/hello
    when: not ansible_check_mode

  - name: This task will ignore errors in check mode
    ansible.builtin.git:
      repo: ssh://git@github.com/mylogin/hello.git
      dest: /home/mylogin/hello
    ignore_errors: "{{ ansible_check_mode }}"
```

Using diff mode

The `--diff` option for ansible-playbook can be used alone or with `--check`. When you run in diff mode, any module that supports diff mode reports the changes made or, if used with `--check`, the changes that would have been made. Diff mode is most common in modules that manipulate files (for example, the template module) but other modules might also show 'before and after' information (for example, the user module).

Diff mode produces a large amount of output, so it is best used when checking a single host at a time. For example:

```
ansible-playbook foo.yml --check --diff --limit foo.example.com
```

New in version 2.4.

Enforcing or preventing diff mode on tasks

Because the `--diff` option can reveal sensitive information, you can disable it for a task by specifying `diff: no`. For example:

```yaml
tasks:
  - name: This task will not report a diff when the file changes
    ansible.builtin.template:
      src: secret.conf.j2
```

(continues on next page)

(continued from previous page)

```
      dest: /etc/secret.conf
      owner: root
      group: root
      mode: '0600'
    diff: false
```

Understanding privilege escalation: become

Ansible uses existing privilege escalation systems to execute tasks with root privileges or with another user's permissions. Because this feature allows you to 'become' another user, different from the user that logged into the machine (remote user), we call it `become`. The `become` keyword uses existing privilege escalation tools like *sudo*, *su*, *pfexec*, *doas*, *pbrun*, *dzdo*, *ksu*, *runas*, *machinectl* and others.

Using become

You can control the use of become with play or task directives, connection variables, or at the command line. If you set privilege escalation properties in multiple ways, review the *general precedence rules* to understand which settings will be used.

A full list of all become plugins that are included in Ansible can be found in the *Plugin List*.

Become directives

You can set the directives that control become at the play or task level. You can override these by setting connection variables, which often differ from one host to another. These variables and directives are independent. For example, setting become_user does not set become.

become
    set to yes to activate privilege escalation.

become_user
    set to user with desired privileges — the user you *become*, NOT the user you login as. Does NOT imply become: true, to allow it to be set at host level. Default value is root.

become_method
    (at play or task level) overrides the default method set in ansible.cfg, set to use any of the *Become plugins*.

become_flags
    (at play or task level) permit the use of specific flags for the tasks or role. One common use is to change the user to nobody when the shell is set to nologin. Added in Ansible 2.2.

For example, to manage a system service (which requires root privileges) when connected as a non-root user, you can use the default value of become_user (root):

```
- name: Ensure the httpd service is running
  service:
    name: httpd
    state: started
  become: true
```

To run a command as the apache user:

```
- name: Run a command as the apache user
  command: somecommand
  become: true
  become_user: apache
```

To do something as the nobody user when the shell is nologin:

```
- name: Run a command as nobody
  command: somecommand
  become: true
  become_method: su
  become_user: nobody
  become_flags: '-s /bin/sh'
```

To specify a password for sudo, run ansible-playbook with --ask-become-pass (-K for short). If you run a playbook utilizing become and the playbook seems to hang, most likely it is stuck at the privilege escalation prompt. Stop it with *CTRL-c*, then execute the playbook with -K and the appropriate password.

Become connection variables

You can define different `become` options for each managed node or group. You can define these variables in inventory or use them as normal variables.

ansible_become
: overrides the `become` directive, decides if privilege escalation is used or not.

ansible_become_method
: which privilege escalation method should be used

ansible_become_user
: set the user you become through privilege escalation; does not imply `ansible_become:  true`

ansible_become_password
: set the privilege escalation password. See *Using encrypted variables and files* for details on how to avoid having secrets in plain text

ansible_common_remote_group
: determines if Ansible should try to `chgrp` its temporary files to a group if `setfacl` and `chown` both fail. See *Risks of becoming an unprivileged user* for more information. Added in version 2.10.

For example, if you want to run all tasks as `root` on a server named `webserver`, but you can only connect as the `manager` user, you could use an inventory entry like this:

```
webserver ansible_user=manager ansible_become=yes
```

Note: The variables defined above are generic for all become plugins but plugin specific ones can also be set instead. Please see the documentation for each plugin for a list of all options the plugin has and how they can be defined. A full list of become plugins in Ansible can be found at *Become plugins*.

Become command-line options

--ask-become-pass, -K ask for privilege escalation password; does not imply become will be used. Note that this password will be used for all hosts.

--become, -b run operations with become (no password implied)

--become-method=BECOME_METHOD privilege escalation method to use (default=sudo), valid choices: [sudo | su | pbrun | pfexec | doas | dzdo | ksu | runas | machinectl]

--become-user=BECOME_USER run operations as this user (default=root), does not imply –become/-b

Risks and limitations of become

Although privilege escalation is mostly intuitive, there are a few limitations on how it works. Users should be aware of these to avoid surprises.

Risks of becoming an unprivileged user

Ansible modules are executed on the remote machine by first substituting the parameters into the module file, then copying the file to the remote machine, and finally executing it there.

Everything is fine if the module file is executed without using `become`, when the `become_user` is root, or when the connection to the remote machine is made as root. In these cases Ansible creates the module file with permissions that only allow reading by the user and root, or only allow reading by the unprivileged user being switched to.

However, when both the connection user and the `become_user` are unprivileged, the module file is written as the user that Ansible connects as (the `remote_user`), but the file needs to be readable by the user Ansible is set to `become`. The details of how Ansible solves this can vary based on platform. However, on POSIX systems, Ansible solves this problem in the following way:

First, if **`setfacl`** is installed and available in the remote `PATH`, and the temporary directory on the remote host is mounted with POSIX.1e filesystem ACL support, Ansible will use POSIX ACLs to share the module file with the second unprivileged user.

Next, if POSIX ACLs are **not** available or **`setfacl`** could not be run, Ansible will attempt to change ownership of the module file using **`chown`** for systems which support doing so as an unprivileged user.

New in Ansible 2.11, at this point, Ansible will try **`chmod +a`** which is a macOS-specific way of setting ACLs on files.

New in Ansible 2.10, if all of the above fails, Ansible will then check the value of the configuration setting `ansible_common_remote_group`. Many systems will allow a given user to change the group ownership of a file to a group the user is in. As a result, if the second unprivileged user (the `become_user`) has a UNIX group in common with the user Ansible is connected as (the `remote_user`), and if `ansible_common_remote_group` is defined to be that group, Ansible can try to change the group ownership of the module file to that group by using **`chgrp`**, thereby likely making it readable to the `become_user`.

At this point, if `ansible_common_remote_group` was defined and a **`chgrp`** was attempted and returned successfully, Ansible assumes (but, importantly, does not check) that the new group ownership is enough and does not fall back further. That is, Ansible **does not check** that the `become_user` does in fact share a group with the `remote_user`; so long as the command exits successfully, Ansible considers the result successful and does not proceed to check `allow_world_readable_tmpfiles` per below.

If `ansible_common_remote_group` is **not** set and the chown above it failed, or if `ansible_common_remote_group` *is* set but the **`chgrp`** (or following group-permissions **`chmod`**) returned a non-successful exit code, Ansible will lastly check the value of `allow_world_readable_tmpfiles`. If this is set, Ansible will place the module file in a world-readable temporary directory, with world-readable permissions to allow the `become_user` (and incidentally any other user on the system) to read the contents of the file. **If any of the parameters passed to the module are sensitive in nature, and you do not trust the remote machines, then this is a potential security risk.**

Once the module is done executing, Ansible deletes the temporary file.

Several ways exist to avoid the above logic flow entirely:

- Use *pipelining*. When pipelining is enabled, Ansible does not save the module to a temporary file on the client. Instead it pipes the module to the remote python interpreter's stdin. Pipelining does not work for python modules involving file transfer (for example: copy, fetch, template), or for non-python modules.
- Avoid becoming an unprivileged user. Temporary files are protected by UNIX file permissions when you `become` root or do not use `become`. In Ansible 2.1 and above, UNIX file permissions are also secure if you make the connection to the managed machine as root and then use `become` to access an unprivileged account.

Warning: Although the Solaris ZFS filesystem has filesystem ACLs, the ACLs are not POSIX.1e filesystem acls (they are NFSv4 ACLs instead). Ansible cannot use these ACLs to manage its temp file permissions so you may have to resort to `allow_world_readable_tmpfiles` if the remote machines use ZFS.

Changed in version 2.1.

Ansible makes it hard to unknowingly use `become` insecurely. Starting in Ansible 2.1, Ansible defaults to issuing an error if it cannot execute securely with `become`. If you cannot use pipelining or POSIX ACLs, must connect as an unprivileged user, must use `become` to execute as a different unprivileged user, and decide that your managed nodes are secure enough for the modules you want to run there to be world readable, you can turn on `allow_world_readable_tmpfiles` in the `ansible.cfg` file. Setting `allow_world_readable_tmpfiles` will change this from an error into a warning and allow the task to run as it did prior to 2.1.

Changed in version 2.10.

Ansible 2.10 introduces the above-mentioned `ansible_common_remote_group` fallback. As mentioned above, if enabled, it is used when `remote_user` and `become_user` are both unprivileged users. Refer to the text above for details on when this fallback happens.

Warning: As mentioned above, if `ansible_common_remote_group` and `allow_world_readable_tmpfiles` are both enabled, it is unlikely that the world-readable fallback will ever trigger, and yet Ansible might still be unable to access the module file. This is because after the group ownership change is successful, Ansible does not fall back any further, and also does not do any check to ensure that the `become_user` is actually a member of the "common group". This is a design decision made by the fact that doing such a check would require another round-trip connection to the remote machine, which is a time-expensive operation. Ansible does, however, emit a warning in this case.

Not supported by all connection plugins

Privilege escalation methods must also be supported by the connection plugin used. Most connection plugins will warn if they do not support become. Some will just ignore it as they always run as root (jail, chroot, and so on).

Only one method may be enabled per host

Methods cannot be chained. You cannot use `sudo /bin/su -` to become a user, you need to have privileges to run the command as that user in sudo or be able to su directly to it (the same for pbrun, pfexec or other supported methods).

Privilege escalation must be general

You cannot limit privilege escalation permissions to certain commands. Ansible does not always use a specific command to do something but runs modules (code) from a temporary file name which changes every time. If you have '/sbin/service' or '/bin/chmod' as the allowed commands this will fail with ansible as those paths won't match with the temporary file that Ansible creates to run the module. If you have security rules that constrain your sudo/pbrun/doas environment to running specific command paths only, use Ansible from a special account that does not have this constraint, or use AWX or the *Red Hat Ansible Automation Platform* to manage indirect access to SSH credentials.

May not access environment variables populated by pamd_systemd

For most Linux distributions using `systemd` as their init, the default methods used by `become` do not open a new "session", in the sense of systemd. Because the `pam_systemd` module will not fully initialize a new session, you might have surprises compared to a normal session opened through ssh: some environment variables set by `pam_systemd`, most notably `XDG_RUNTIME_DIR`, are not populated for the new user and instead inherited or just emptied.

This might cause trouble when trying to invoke systemd commands that depend on `XDG_RUNTIME_DIR` to access the bus:

```
$ echo $XDG_RUNTIME_DIR

$ systemctl --user status
Failed to connect to bus: Permission denied
```

To force `become` to open a new systemd session that goes through `pam_systemd`, you can use `become_method: machinectl`.

For more information, see this systemd issue.

Resolving Temporary File Error Messsages

- Failed to set permissions on the temporary files Ansible needs to create when becoming an unprivileged user"
- This error can be resolved by installing the package that provides the `setfacl` command. (This is frequently the `acl` package but check your OS documentation.)

Become and network automation

As of version 2.6, Ansible supports `become` for privilege escalation (entering `enable` mode or privileged EXEC mode) on all Ansible-maintained network platforms that support `enable` mode. Using `become` replaces the `authorize` and `auth_pass` options in a `provider` dictionary.

You must set the connection type to either `connection: ansible.netcommon.network_cli` or `connection: ansible.netcommon.httpapi` to use `become` for privilege escalation on network devices. Check the *Platform Options* documentation for details.

You can use escalated privileges on only the specific tasks that need them, on an entire play, or on all plays. Adding `become: true` and `become_method: enable` instructs Ansible to enter `enable` mode before executing the task, play, or playbook where those parameters are set.

If you see this error message, the task that generated it requires `enable` mode to succeed:

```
Invalid input (privileged mode required)
```

To set `enable` mode for a specific task, add `become` at the task level:

```
- name: Gather facts (eos)
  arista.eos.eos_facts:
    gather_subset:
      - "!hardware"
  become: true
  become_method: enable
```

To set enable mode for all tasks in a single play, add `become` at the play level:

```
- hosts: eos-switches
  become: true
  become_method: enable
  tasks:
    - name: Gather facts (eos)
      arista.eos.eos_facts:
        gather_subset:
          - "!hardware"
```

Setting enable mode for all tasks

Often you wish for all tasks in all plays to run using privilege mode, that is best achieved by using `group_vars`:

group_vars/eos.yml

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: arista.eos.eos
ansible_user: myuser
ansible_become: true
ansible_become_method: enable
```

Passwords for enable mode

If you need a password to enter `enable` mode, you can specify it in one of two ways:

- providing the *--ask-become-pass* command line option
- setting the `ansible_become_password` connection variable

Warning: As a reminder passwords should never be stored in plain text. For information on encrypting your passwords and other secrets with Ansible Vault, see *Ansible Vault*.

authorize and auth_pass

Ansible still supports `enable` mode with `connection: local` for legacy network playbooks. To enter `enable` mode with `connection: local`, use the module options `authorize` and `auth_pass`:

```
- hosts: eos-switches
  ansible_connection: local
  tasks:
    - name: Gather facts (eos)
      eos_facts:
        gather_subset:
          - "!hardware"
      provider:
        authorize: true
        auth_pass: " {{ secret_auth_pass }}"
```

We recommend updating your playbooks to use `become` for network-device `enable` mode consistently. The use of `authorize` and of `provider` dictionaries will be deprecated in future. Check the *Platform Options* documentation for details.

Become and Windows

Since Ansible 2.3, `become` can be used on Windows hosts through the `runas` method. Become on Windows uses the same inventory setup and invocation arguments as `become` on a non-Windows host, so the setup and variable names are the same as what is defined in this document with the exception of `become_user`. As there is no sensible default for `become_user` on Windows it is required when using `become`. See ansible.builtin.runas become plugin for details.

While `become` can be used to assume the identity of another user, there are other uses for it with Windows hosts. One important use is to bypass some of the limitations that are imposed when running on WinRM, such as constrained network delegation or accessing forbidden system calls like the WUA API. You can use `become` with the same user as `ansible_user` to bypass these limitations and run commands that are not normally accessible in a WinRM session.

Note: On Windows you cannot connect with an underprivileged account and use become to elevate your rights. Become can only be used if your connection account is already an Administrator of the target host.

Administrative rights

Many tasks in Windows require administrative privileges to complete. When using the `runas` become method, Ansible will attempt to run the module with the full privileges that are available to the become user. If it fails to elevate the user token, it will continue to use the limited token during execution.

A user must have the `SeDebugPrivilege` to run a become process with elevated privileges. This privilege is assigned to Administrators by default. If the debug privilege is not available, the become process will run with a limited set of privileges and groups.

To determine the type of token that Ansible was able to get, run the following task:

```
- Check my user name
  ansible.windows.win_whoami:
  become: true
```

The output will look something similar to the below:

```
ok: [windows] => {
    "account": {
        "account_name": "vagrant-domain",
        "domain_name": "DOMAIN",
        "sid": "S-1-5-21-3088887838-4058132883-1884671576-1105",
        "type": "User"
    },
    "authentication_package": "Kerberos",
    "changed": false,
    "dns_domain_name": "DOMAIN.LOCAL",
    "groups": [
        {
            "account_name": "Administrators",
            "attributes": [
                "Mandatory",
```

(continues on next page)

(continued from previous page)

```
                "Enabled by default",
                "Enabled",
                "Owner"
            ],
            "domain_name": "BUILTIN",
            "sid": "S-1-5-32-544",
            "type": "Alias"
        },
        {
            "account_name": "INTERACTIVE",
            "attributes": [
                "Mandatory",
                "Enabled by default",
                "Enabled"
            ],
            "domain_name": "NT AUTHORITY",
            "sid": "S-1-5-4",
            "type": "WellKnownGroup"
        },
    ],
    "impersonation_level": "SecurityAnonymous",
    "label": {
        "account_name": "High Mandatory Level",
        "domain_name": "Mandatory Label",
        "sid": "S-1-16-12288",
        "type": "Label"
    },
    "login_domain": "DOMAIN",
    "login_time": "2018-11-18T20:35:01.9696884+00:00",
    "logon_id": 114196830,
    "logon_server": "DC01",
    "logon_type": "Interactive",
    "privileges": {
        "SeBackupPrivilege": "disabled",
        "SeChangeNotifyPrivilege": "enabled-by-default",
        "SeCreateGlobalPrivilege": "enabled-by-default",
        "SeCreatePagefilePrivilege": "disabled",
        "SeCreateSymbolicLinkPrivilege": "disabled",
        "SeDebugPrivilege": "enabled",
        "SeDelegateSessionUserImpersonatePrivilege": "disabled",
        "SeImpersonatePrivilege": "enabled-by-default",
        "SeIncreaseBasePriorityPrivilege": "disabled",
        "SeIncreaseQuotaPrivilege": "disabled",
        "SeIncreaseWorkingSetPrivilege": "disabled",
        "SeLoadDriverPrivilege": "disabled",
        "SeManageVolumePrivilege": "disabled",
        "SeProfileSingleProcessPrivilege": "disabled",
        "SeRemoteShutdownPrivilege": "disabled",
        "SeRestorePrivilege": "disabled",
        "SeSecurityPrivilege": "disabled",
        "SeShutdownPrivilege": "disabled",
        "SeSystemEnvironmentPrivilege": "disabled",
```

(continues on next page)

(continued from previous page)

```
        "SeSystemProfilePrivilege": "disabled",
        "SeSystemtimePrivilege": "disabled",
        "SeTakeOwnershipPrivilege": "disabled",
        "SeTimeZonePrivilege": "disabled",
        "SeUndockPrivilege": "disabled"
    },
    "rights": [
        "SeNetworkLogonRight",
        "SeBatchLogonRight",
        "SeInteractiveLogonRight",
        "SeRemoteInteractiveLogonRight"
    ],
    "token_type": "TokenPrimary",
    "upn": "vagrant-domain@DOMAIN.LOCAL",
    "user_flags": []
}
```

Under the `label` key, the `account_name` entry determines whether the user has Administrative rights. Here are the labels that can be returned and what they represent:

- `Medium`: Ansible failed to get an elevated token and ran under a limited token. Only a subset of the privileges assigned to user are available during the module execution and the user does not have administrative rights.
- `High`: An elevated token was used and all the privileges assigned to the user are available during the module execution.
- `System`: The `NT AUTHORITY\System` account is used and has the highest level of privileges available.

The output will also show the list of privileges that have been granted to the user. When the privilege value is `disabled`, the privilege is assigned to the logon token but has not been enabled. In most scenarios these privileges are automatically enabled when required.

If running on a version of Ansible that is older than 2.5 or the normal `runas` escalation process fails, an elevated token can be retrieved by:

- Set the `become_user` to `System` which has full control over the operating system.
- Grant `SeTcbPrivilege` to the user Ansible connects with on WinRM. `SeTcbPrivilege` is a high-level privilege that grants full control over the operating system. No user is given this privilege by default, and care should be taken if you grant this privilege to a user or group. For more information on this privilege, please see Act as part of the operating system. You can use the below task to set this privilege on a Windows host:

  ```
  - name: grant the ansible user the SeTcbPrivilege right
    ansible.windows.win_user_right:
      name: SeTcbPrivilege
      users: '{{ansible_user}}'
      action: add
  ```

- Turn UAC off on the host and reboot before trying to become the user. UAC is a security protocol that is designed to run accounts with the `least privilege` principle. You can turn UAC off by running the following tasks:

  ```
  - name: turn UAC off
    win_regedit:
      path: HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system
      name: EnableLUA
      data: 0
  ```

(continues on next page)

(continued from previous page)

```
    type: dword
    state: present
  register: uac_result

- name: reboot after disabling UAC
  win_reboot:
  when: uac_result is changed
```

Note: Granting the `SeTcbPrivilege` or turning UAC off can cause Windows security vulnerabilities and care should be given if these steps are taken.

Local service accounts

Prior to Ansible version 2.5, `become` only worked on Windows with a local or domain user account. Local service accounts like `System` or `NetworkService` could not be used as `become_user` in these older versions. This restriction has been lifted since the 2.5 release of Ansible. The three service accounts that can be set under `become_user` are:

- System
- NetworkService
- LocalService

Because local service accounts do not have passwords, the `ansible_become_password` parameter is not required and is ignored if specified.

Become without setting a password

As of Ansible 2.8, `become` can be used to become a Windows local or domain account without requiring a password for that account. For this method to work, the following requirements must be met:

- The connection user has the `SeDebugPrivilege` privilege assigned
- The connection user is part of the `BUILTIN\Administrators` group
- The `become_user` has either the `SeBatchLogonRight` or `SeNetworkLogonRight` user right

Using become without a password is achieved in one of two different methods:

- Duplicating an existing logon session's token if the account is already logged on
- Using S4U to generate a logon token that is valid on the remote host only

In the first scenario, the become process is spawned from another logon of that user account. This could be an existing RDP logon, console logon, but this is not guaranteed to occur all the time. This is similar to the `Run only when user is logged on` option for a Scheduled Task.

In the case where another logon of the become account does not exist, S4U is used to create a new logon and run the module through that. This is similar to the `Run whether user is logged on or not` with the `Do not store password` option for a Scheduled Task. In this scenario, the become process will not be able to access any network resources like a normal WinRM process.

To make a distinction between using become with no password and becoming an account that has no password make sure to keep `ansible_become_password` as undefined or set `ansible_become_password:`.

Note: Because there are no guarantees an existing token will exist for a user when Ansible runs, there's a high change the become process will only have access to local resources. Use become with a password if the task needs to access network resources

Accounts without a password

Warning: As a general security best practice, you should avoid allowing accounts without passwords.

Ansible can be used to become a Windows account that does not have a password (like the `Guest` account). To become an account without a password, set up the variables like normal but set `ansible_become_password: ''`.

Before become can work on an account like this, the local policy Accounts: Limit local account use of blank passwords to console logon only must be disabled. This can either be done through a Group Policy Object (GPO) or with this Ansible task:

```
- name: allow blank password on become
  ansible.windows.win_regedit:
    path: HKLM:\SYSTEM\CurrentControlSet\Control\Lsa
    name: LimitBlankPasswordUse
    data: 0
    type: dword
    state: present
```

Note: This is only for accounts that do not have a password. You still need to set the account's password under `ansible_become_password` if the become_user has a password.

Become flags for Windows

Ansible 2.5 added the `become_flags` parameter to the `runas` become method. This parameter can be set using the `become_flags` task directive or set in Ansible's configuration using `ansible_become_flags`. The two valid values that are initially supported for this parameter are `logon_type` and `logon_flags`.

Note: These flags should only be set when becoming a normal user account, not a local service account like LocalSystem.

The key `logon_type` sets the type of logon operation to perform. The value can be set to one of the following:

- `interactive`: The default logon type. The process will be run under a context that is the same as when running a process locally. This bypasses all WinRM restrictions and is the recommended method to use.
- `batch`: Runs the process under a batch context that is similar to a scheduled task with a password set. This should bypass most WinRM restrictions and is useful if the `become_user` is not allowed to log on interactively.
- `new_credentials`: Runs under the same credentials as the calling user, but outbound connections are run under the context of the `become_user` and `become_password`, similar to `runas.exe /netonly`. The `logon_flags` flag should also be set to `netcredentials_only`. Use this flag if the process needs to access a network resource (like an SMB share) using a different set of credentials.

- network: Runs the process under a network context without any cached credentials. This results in the same type of logon session as running a normal WinRM process without credential delegation, and operates under the same restrictions.
- network_cleartext: Like the network logon type, but instead caches the credentials so it can access network resources. This is the same type of logon session as running a normal WinRM process with credential delegation.

For more information, see dwLogonType.

The logon_flags key specifies how Windows will log the user on when creating the new process. The value can be set to none or multiple of the following:

- with_profile: The default logon flag set. The process will load the user's profile in the HKEY_USERS registry key to HKEY_CURRENT_USER.
- netcredentials_only: The process will use the same token as the caller but will use the become_user and become_password when accessing a remote resource. This is useful in inter-domain scenarios where there is no trust relationship, and should be used with the new_credentials logon_type.

By default logon_flags=with_profile is set, if the profile should not be loaded set logon_flags= or if the profile should be loaded with netcredentials_only, set logon_flags=with_profile,netcredentials_only.

For more information, see dwLogonFlags.

Here are some examples of how to use become_flags with Windows tasks:

```yaml
- name: copy a file from a fileshare with custom credentials
  ansible.windows.win_copy:
    src: \\server\share\data\file.txt
    dest: C:\temp\file.txt
    remote_src: true
  vars:
    ansible_become: true
    ansible_become_method: runas
    ansible_become_user: DOMAIN\user
    ansible_become_password: Password01
    ansible_become_flags: logon_type=new_credentials logon_flags=netcredentials_only

- name: run a command under a batch logon
  ansible.windows.win_whoami:
  become: true
  become_flags: logon_type=batch

- name: run a command and not load the user profile
  ansible.windows.win_whomai:
  become: true
  become_flags: logon_flags=
```

Limitations of become on Windows

- Running a task with `async` and `become` on Windows Server 2008, 2008 R2 and Windows 7 only works when using Ansible 2.7 or newer.
- By default, the become user logs on with an interactive session, so it must have the right to do so on the Windows host. If it does not inherit the `SeAllowLogOnLocally` privilege or inherits the `SeDenyLogOnLocally` privilege, the become process will fail. Either add the privilege or set the `logon_type` flag to change the logon type used.
- Prior to Ansible version 2.3, become only worked when `ansible_winrm_transport` was either `basic` or `credssp`. This restriction has been lifted since the 2.4 release of Ansible for all hosts except Windows Server 2008 (non R2 version).
- The Secondary Logon service `seclogon` must be running to use `ansible_become_method: runas`
- The connection user must already be an Administrator on the Windows host to use `runas`. The target become user does not need to be an Administrator though.

See also:

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

Tags

If you have a large playbook, it may be useful to run only specific parts of it instead of running the entire playbook. You can do this with Ansible tags. Using tags to execute or skip selected tasks is a two-step process:

1. Add tags to your tasks, either individually or with tag inheritance from a block, play, role, or import.
2. Select or skip tags when you run your playbook.

Adding tags with the tags keyword

You can add tags to a single task or include. You can also add tags to multiple tasks by defining them at the level of a block, play, role, or import. The keyword `tags` addresses all these use cases. The `tags` keyword always defines tags and adds them to tasks; it does not select or skip tasks for execution. You can only select or skip tasks based on tags at the command line when you run a playbook. See *Selecting or skipping tags when you run a playbook* for more details.

Adding tags to individual tasks

At the simplest level, you can apply one or more tags to an individual task. You can add tags to tasks in playbooks, in task files, or within a role. Here is an example that tags two tasks with different tags:

```
tasks:
- name: Install the servers
  ansible.builtin.yum:
    name:
    - httpd
    - memcached
    state: present
  tags:
  - packages
  - webservers

- name: Configure the service
  ansible.builtin.template:
    src: templates/src.j2
    dest: /etc/foo.conf
  tags:
  - configuration
```

You can apply the same tag to more than one individual task. This example tags several tasks with the same tag, "ntp":

```
---
# file: roles/common/tasks/main.yml

- name: Install ntp
  ansible.builtin.yum:
    name: ntp
    state: present
  tags: ntp

- name: Configure ntp
  ansible.builtin.template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
  - restart ntpd
  tags: ntp

- name: Enable and run ntpd
  ansible.builtin.service:
    name: ntpd
```

(continues on next page)

(continued from previous page)

```
    state: started
    enabled: true
  tags: ntp

- name: Install NFS utils
  ansible.builtin.yum:
    name:
    - nfs-utils
    - nfs-util-lib
    state: present
  tags: filesharing
```

If you ran these four tasks in a playbook with `--tags ntp`, Ansible would run the three tasks tagged `ntp` and skip the one task that does not have that tag.

Adding tags to includes

You can apply tags to dynamic includes in a playbook. As with tags on an individual task, tags on an `include_*` task apply only to the include itself, not to any tasks within the included file or role. If you add `mytag` to a dynamic include, then run that playbook with `--tags mytag`, Ansible runs the include itself, runs any tasks within the included file or role tagged with `mytag`, and skips any tasks within the included file or role without that tag. See *Selectively running tagged tasks in re-usable files* for more details.

You add tags to includes the same way you add tags to any other task:

```
---
# file: roles/common/tasks/main.yml

- name: Dynamic re-use of database tasks
  include_tasks: db.yml
  tags: db
```

You can add a tag only to the dynamic include of a role. In this example, the `foo` tag will *not* apply to tasks inside the `bar` role:

```
---
- hosts: webservers
  tasks:
    - name: Include the bar role
      include_role:
        name: bar
      tags:
        - foo
```

With plays, blocks, the `role` keyword, and static imports, Ansible applies tag inheritance, adding the tags you define to every task inside the play, block, role, or imported file. However, tag inheritance does *not* apply to dynamic re-use with `include_role` and `include_tasks`. With dynamic re-use (includes), the tags you define apply only to the include itself. If you need tag inheritance, use a static import. If you cannot use an import because the rest of your playbook uses includes, see *Tag inheritance for includes: blocks and the apply keyword* for ways to work around this behavior.

Tag inheritance: adding tags to multiple tasks

If you want to apply the same tag or tags to multiple tasks without adding a `tags` line to every task, you can define the tags at the level of your play or block, or when you add a role or import a file. Ansible applies the tags down the dependency chain to all child tasks. With roles and imports, Ansible appends the tags set by the `roles` section or import to any tags set on individual tasks or blocks within the role or imported file. This is called tag inheritance. Tag inheritance is convenient, because you do not have to tag every task. However, the tags still apply to the tasks individually.

Adding tags to blocks

If you want to apply a tag to many, but not all, of the tasks in your play, use a *block* and define the tags at that level. For example, we could edit the NTP example shown above to use a block:

```yaml
# myrole/tasks/main.yml
- name: ntp tasks
  tags: ntp
  block:
  - name: Install ntp
    ansible.builtin.yum:
      name: ntp
      state: present

  - name: Configure ntp
    ansible.builtin.template:
      src: ntp.conf.j2
      dest: /etc/ntp.conf
    notify:
    - restart ntpd

  - name: Enable and run ntpd
    ansible.builtin.service:
      name: ntpd
      state: started
      enabled: true

- name: Install NFS utils
  ansible.builtin.yum:
    name:
    - nfs-utils
    - nfs-util-lib
    state: present
  tags: filesharing
```

Adding tags to plays

If all the tasks in a play should get the same tag, you can add the tag at the level of the play. For example, if you had a play with only the NTP tasks, you could tag the entire play:

```
- hosts: all
  tags: ntp
  tasks:
  - name: Install ntp
    ansible.builtin.yum:
      name: ntp
      state: present

  - name: Configure ntp
    ansible.builtin.template:
      src: ntp.conf.j2
      dest: /etc/ntp.conf
    notify:
    - restart ntpd

  - name: Enable and run ntpd
    ansible.builtin.service:
      name: ntpd
      state: started
      enabled: true

- hosts: fileservers
  tags: filesharing
  tasks:
  ...
```

Adding tags to roles

There are three ways to add tags to roles:

1. Add the same tag or tags to all tasks in the role by setting tags under `roles`. See examples in this section.
2. Add the same tag or tags to all tasks in the role by setting tags on a static `import_role` in your playbook. See examples in *Adding tags to imports*.
3. Add a tag or tags to individual tasks or blocks within the role itself. This is the only approach that allows you to select or skip some tasks within the role. To select or skip tasks within the role, you must have tags set on individual tasks or blocks, use the dynamic `include_role` in your playbook, and add the same tag or tags to the include. When you use this approach, and then run your playbook with `--tags foo`, Ansible runs the include itself plus any tasks in the role that also have the tag `foo`. See *Adding tags to includes* for details.

When you incorporate a role in your playbook statically with the `roles` keyword, Ansible adds any tags you define to all the tasks in the role. For example:

```
roles:
  - role: webserver
    vars:
      port: 5000
    tags: [ web, foo ]
```

or:

```
---
- hosts: webservers
  roles:
    - role: foo
      tags:
        - bar
        - baz
    # using YAML shorthand, this is equivalent to:
    # - { role: foo, tags: ["bar", "baz"] }
```

Adding tags to imports

You can also apply a tag or tags to all the tasks imported by the static `import_role` and `import_tasks` statements:

```
---
- hosts: webservers
  tasks:
    - name: Import the foo role
      import_role:
        name: foo
      tags:
        - bar
        - baz

    - name: Import tasks from foo.yml
      import_tasks: foo.yml
      tags: [ web, foo ]
```

Tag inheritance for includes: blocks and the `apply` keyword

By default, Ansible does not apply *tag inheritance* to dynamic re-use with `include_role` and `include_tasks`. If you add tags to an include, they apply only to the include itself, not to any tasks in the included file or role. This allows you to execute selected tasks within a role or task file - see *Selectively running tagged tasks in re-usable files* when you run your playbook.

If you want tag inheritance, you probably want to use imports. However, using both includes and imports in a single playbook can lead to difficult-to-diagnose bugs. For this reason, if your playbook uses `include_*` to re-use roles or tasks, and you need tag inheritance on one include, Ansible offers two workarounds. You can use the `apply` keyword:

```
- name: Apply the db tag to the include and to all tasks in db.yml
  include_tasks:
    file: db.yml
    # adds 'db' tag to tasks within db.yml
    apply:
      tags: db
  # adds 'db' tag to this 'include_tasks' itself
  tags: db
```

Or you can use a block:

```
- block:
  - name: Include tasks from db.yml
    include_tasks: db.yml
  tags: db
```

Special tags: always and never

Ansible reserves two tag names for special behavior: always and never. If you assign the `always` tag to a task or play, Ansible will always run that task or play, unless you specifically skip it (`--skip-tags always`).

For example:

```
tasks:
- name: Print a message
  ansible.builtin.debug:
    msg: "Always runs"
  tags:
  - always

- name: Print a message
  ansible.builtin.debug:
    msg: "runs when you use tag1"
  tags:
  - tag1
```

Warning:

- Fact gathering is tagged with 'always' by default. It is only skipped if you apply a tag to the play and then use a different tag in `--tags` or the same tag in `--skip-tags`.

Warning:

- The role argument specification validation task is tagged with 'always' by default. This validation will be skipped if you use `--skip-tags always`.

New in version 2.5.

If you assign the `never` tag to a task or play, Ansible will skip that task or play unless you specifically request it (`--tags never`).

For example:

```
tasks:
  - name: Run the rarely-used debug task
    ansible.builtin.debug:
     msg: '{{ showmevar }}'
    tags: [ never, debug ]
```

The rarely-used debug task in the example above only runs when you specifically request the `debug` or `never` tags.

Selecting or skipping tags when you run a playbook

Once you have added tags to your tasks, includes, blocks, plays, roles, and imports, you can selectively execute or skip tasks based on their tags when you run *ansible-playbook*. Ansible runs or skips all tasks with tags that match the tags you pass at the command line. If you have added a tag at the block or play level, with `roles`, or with an import, that tag applies to every task within the block, play, role, or imported role or file. If you have a role with lots of tags and you want to call subsets of the role at different times, either *use it with dynamic includes*, or split the role into multiple roles.

ansible-playbook offers five tag-related command-line options:

- `--tags all` - run all tasks, ignore tags (default behavior)
- `--tags [tag1, tag2]` - run only tasks with either the tag `tag1` or the tag `tag2`
- `--skip-tags [tag3, tag4]` - run all tasks except those with either the tag `tag3` or the tag `tag4`
- `--tags tagged` - run only tasks with at least one tag
- `--tags untagged` - run only tasks with no tags

For example, to run only tasks and blocks tagged either `configuration` or `packages` in a very long playbook:

```
ansible-playbook example.yml --tags "configuration,packages"
```

To run all tasks except those tagged `packages`:

```
ansible-playbook example.yml --skip-tags "packages"
```

To run all tasks, even those excluded because are tagged `never`:

```
ansible-playbook example.yml --tags "all,never"
```

Previewing the results of using tags

When you run a role or playbook, you might not know or remember which tasks have which tags, or which tags exist at all. Ansible offers two command-line flags for *ansible-playbook* that help you manage tagged playbooks:

- `--list-tags` - generate a list of available tags
- `--list-tasks` - when used with `--tags tagname` or `--skip-tags tagname`, generate a preview of tagged tasks

For example, if you do not know whether the tag for configuration tasks is `config` or `conf` in a playbook, role, or tasks file, you can display all available tags without running any tasks:

```
ansible-playbook example.yml --list-tags
```

If you do not know which tasks have the tags `configuration` and `packages`, you can pass those tags and add `--list-tasks`. Ansible lists the tasks but does not execute any of them.

```
ansible-playbook example.yml --tags "configuration,packages" --list-tasks
```

These command-line flags have one limitation: they cannot show tags or tasks within dynamically included files or roles. See *Comparing includes and imports: dynamic and static re-use* for more information on differences between static imports and dynamic includes.

Selectively running tagged tasks in re-usable files

If you have a role or a tasks file with tags defined at the task or block level, you can selectively run or skip those tagged tasks in a playbook if you use a dynamic include instead of a static import. You must use the same tag on the included tasks and on the include statement itself. For example you might create a file with some tagged and some untagged tasks:

```
# mixed.yml
tasks:
- name: Run the task with no tags
  ansible.builtin.debug:
    msg: this task has no tags

- name: Run the tagged task
  ansible.builtin.debug:
    msg: this task is tagged with mytag
  tags: mytag

- block:
  - name: Run the first block task with mytag
    ...
  - name: Run the second block task with mytag
    ...
  tags:
  - mytag
```

And you might include the tasks file above in a playbook:

```
# myplaybook.yml
- hosts: all
  tasks:
  - name: Run tasks from mixed.yml
    include_tasks:
      name: mixed.yml
    tags: mytag
```

When you run the playbook with `ansible-playbook -i hosts myplaybook.yml --tags "mytag"`, Ansible skips the task with no tags, runs the tagged individual task, and runs the two tasks in the block.

Configuring tags globally

If you run or skip certain tags by default, you can use the *TAGS_RUN* and *TAGS_SKIP* options in Ansible configuration to set those defaults.

See also:

Ansible playbooks
An introduction to playbooks

Roles
Playbook organization by roles

User Mailing List
Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Executing playbooks for troubleshooting

When you are testing new plays or debugging playbooks, you may need to run the same play multiple times. To make this more efficient, Ansible offers two alternative ways to execute a playbook: start-at-task and step mode.

start-at-task

To start executing your playbook at a particular task (usually the task that failed on the previous run), use the `--start-at-task` option.

```
ansible-playbook playbook.yml --start-at-task="install packages"
```

In this example, Ansible starts executing your playbook at a task named "install packages". This feature does not work with tasks inside dynamically re-used roles or tasks (`include_*`), see *Comparing includes and imports: dynamic and static re-use*.

Step mode

To execute a playbook interactively, use `--step`.

```
ansible-playbook playbook.yml --step
```

With this option, Ansible stops on each task, and asks if it should execute that task. For example, if you have a task called "configure ssh", the playbook run will stop and ask.

```
Perform task: configure ssh (y/n/c):
```

Answer "y" to execute the task, answer "n" to skip the task, and answer "c" to exit step mode, executing all remaining tasks without asking.

See also:

Ansible playbooks
    An introduction to playbooks

Debugging tasks
    Using the Ansible debugger

Debugging tasks

Ansible offers a task debugger so you can fix errors during execution instead of editing your playbook and running it again to see if your change worked. You have access to all of the features of the debugger in the context of the task. You can check or set the value of variables, update module arguments, and re-run the task with the new variables and arguments. The debugger lets you resolve the cause of the failure and continue with playbook execution.

Enabling the debugger

The debugger is not enabled by default. If you want to invoke the debugger during playbook execution, you must enable it first.

Use one of these three methods to enable the debugger:

- with the debugger keyword
- in configuration or an environment variable, or
- as a strategy

Enabling the debugger with the `debugger` keyword

New in version 2.5.

You can use the `debugger` keyword to enable (or disable) the debugger for a specific play, role, block, or task. This option is especially useful when developing or extending playbooks, plays, and roles. You can enable the debugger on new or updated tasks. If they fail, you can fix the errors efficiently. The `debugger` keyword accepts five values:

Value	Result
always	Always invoke the debugger, regardless of the outcome
never	Never invoke the debugger, regardless of the outcome
on_failed	Only invoke the debugger if a task fails
on_unreachable	Only invoke the debugger if a host is unreachable
on_skipped	Only invoke the debugger if the task is skipped

When you use the `debugger` keyword, the value you specify overrides any global configuration to enable or disable the debugger. If you define `debugger` at multiple levels, such as in a role and in a task, Ansible honors the most granular definition. The definition at the play or role level applies to all blocks and tasks within that play or role, unless they specify a different value. The definition at the block level overrides the definition at the play or role level, and applies to all tasks within that block, unless they specify a different value. The definition at the task level always applies to the task; it overrides the definitions at the block, play, or role level.

Examples of using the `debugger` keyword

Example of setting the `debugger` keyword on a task:

```
- name: Execute a command
  ansible.builtin.command: "false"
  debugger: on_failed
```

Example of setting the `debugger` keyword on a play:

```
- name: My play
  hosts: all
  debugger: on_skipped
  tasks:
    - name: Execute a command
      ansible.builtin.command: "true"
      when: False
```

Example of setting the `debugger` keyword at multiple levels:

```
- name: Play
  hosts: all
  debugger: never
  tasks:
    - name: Execute a command
      ansible.builtin.command: "false"
      debugger: on_failed
```

In this example, the debugger is set to `never` at the play level and to `on_failed` at the task level. If the task fails, Ansible invokes the debugger, because the definition on the task overrides the definition on its parent play.

Enabling the debugger in configuration or an environment variable

New in version 2.5.

You can enable the task debugger globally with a setting in ansible.cfg or with an environment variable. The only options are `True` or `False`. If you set the configuration option or environment variable to `True`, Ansible runs the debugger on failed tasks by default.

To enable the task debugger from ansible.cfg, add this setting to the defaults section:

```
[defaults]
enable_task_debugger = True
```

To enable the task debugger with an environment variable, pass the variable when you run your playbook:

```
ANSIBLE_ENABLE_TASK_DEBUGGER=True ansible-playbook -i hosts site.yml
```

When you enable the debugger globally, every failed task invokes the debugger, unless the role, play, block, or task explicitly disables the debugger. If you need more granular control over what conditions trigger the debugger, use the `debugger` keyword.

Enabling the debugger as a strategy

If you are running legacy playbooks or roles, you may see the debugger enabled as a *strategy*. You can do this at the play level, in ansible.cfg, or with the environment variable `ANSIBLE_STRATEGY=debug`. For example:

```
- hosts: test
  strategy: debug
  tasks:
  ...
```

Or in ansible.cfg:

```
[defaults]
strategy = debug
```

Note: This backwards-compatible method, which matches Ansible versions before 2.5, may be removed in a future release.

Resolving errors in the debugger

After Ansible invokes the debugger, you can use the seven *debugger commands* to resolve the error that Ansible encountered. Consider this example playbook, which defines the `var1` variable but uses the undefined `wrong_var` variable in a task by mistake.

```
- hosts: test
  debugger: on_failed
  gather_facts: false
  vars:
    var1: value1
  tasks:
    - name: Use a wrong variable
      ansible.builtin.ping: data={{ wrong_var }}
```

If you run this playbook, Ansible invokes the debugger when the task fails. From the debug prompt, you can change the module arguments or the variables and run the task again.

```
PLAY ***************************************************************************

TASK [wrong variable] **********************************************************
fatal: [192.0.2.10]: FAILED! => {"failed": true, "msg": "ERROR! 'wrong_var' is undefined
↪"}
Debugger invoked
[192.0.2.10] TASK: wrong variable (debug)> p result._result
{'failed': True,
 'msg': 'The task includes an option with an undefined variable. The error '
        "was: 'wrong_var' is undefined\n"
        '\n'
        'The error appears to have been in '
        "'playbooks/debugger.yml': line 7, "
        'column 7, but may\n'
        'be elsewhere in the file depending on the exact syntax problem.\n'
```

(continues on next page)

(continued from previous page)

```
        '\n'
        'The offending line appears to be:\n'
        '\n'
        '  tasks:\n'
        '    - name: wrong variable\n'
        '      ^ here\n'}
[192.0.2.10] TASK: wrong variable (debug)> p task.args
{u'data': u'{{ wrong_var }}'}
[192.0.2.10] TASK: wrong variable (debug)> task.args['data'] = '{{ var1 }}'
[192.0.2.10] TASK: wrong variable (debug)> p task.args
{u'data': '{{ var1 }}'}
[192.0.2.10] TASK: wrong variable (debug)> redo
ok: [192.0.2.10]

PLAY RECAP *********************************************************************
192.0.2.10               : ok=1    changed=0    unreachable=0    failed=0
```

Changing the task arguments in the debugger to use `var1` instead of `wrong_var` makes the task run successfully.

Available debug commands

You can use these seven commands at the debug prompt:

Command	Shortcut	Action
print	p	Print information about the task
task.args[*key*] = *value*	no shortcut	Update module arguments
task_vars[*key*] = *value*	no shortcut	Update task variables (you must `update_task` next)
update_task	u	Recreate a task with updated task variables
redo	r	Run the task again
continue	c	Continue executing, starting with the next task
quit	q	Quit the debugger

For more details, see the individual descriptions and examples below.

Print command

`print *task/task.args/task_vars/host/result*` prints information about the task.

```
[192.0.2.10] TASK: install package (debug)> p task
TASK: install package
[192.0.2.10] TASK: install package (debug)> p task.args
{u'name': u'{{ pkg_name }}'}
[192.0.2.10] TASK: install package (debug)> p task_vars
{u'ansible_all_ipv4_addresses': [u'192.0.2.10'],
 u'ansible_architecture': u'x86_64',
 ...
}
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
u'bash'
```

(continues on next page)

(continued from previous page)

```
[192.0.2.10] TASK: install package (debug)> p host
192.0.2.10
[192.0.2.10] TASK: install package (debug)> p result._result
{'_ansible_no_log': False,
 'changed': False,
 u'failed': True,
 ...
 u'msg': u"No package matching 'not_exist' is available"}
```

Update args command

`task.args[*key*] = *value*` updates a module argument. This sample playbook has an invalid package name.

```yaml
- hosts: test
  strategy: debug
  gather_facts: true
  vars:
    pkg_name: not_exist
  tasks:
    - name: Install a package
      ansible.builtin.apt: name={{ pkg_name }}
```

When you run the playbook, the invalid package name triggers an error, and Ansible invokes the debugger. You can fix the package name by viewing, then updating the module argument.

```
[192.0.2.10] TASK: install package (debug)> p task.args
{u'name': u'{{ pkg_name }}'}
[192.0.2.10] TASK: install package (debug)> task.args['name'] = 'bash'
[192.0.2.10] TASK: install package (debug)> p task.args
{u'name': 'bash'}
[192.0.2.10] TASK: install package (debug)> redo
```

After you update the module argument, use `redo` to run the task again with the new args.

Update vars command

`task_vars[*key*] = *value*` updates the `task_vars`. You could fix the playbook above by viewing, then updating the task variables instead of the module args.

```
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
u'not_exist'
[192.0.2.10] TASK: install package (debug)> task_vars['pkg_name'] = 'bash'
[192.0.2.10] TASK: install package (debug)> p task_vars['pkg_name']
'bash'
[192.0.2.10] TASK: install package (debug)> update_task
[192.0.2.10] TASK: install package (debug)> redo
```

After you update the task variables, you must use `update_task` to load the new variables before using `redo` to run the task again.

Note: In 2.5 this was updated from `vars` to `task_vars` to avoid conflicts with the `vars()` python function.

Update task command

New in version 2.8.

`u` or `update_task` recreates the task from the original task data structure and templates with updated task variables. See the entry *Update vars command* for an example of use.

Redo command

`r` or `redo` runs the task again.

Continue command

`c` or `continue` continues executing, starting with the next task.

Quit command

`q` or `quit` quits the debugger. The playbook execution is aborted.

How the debugger interacts with the free strategy

With the default `linear` strategy enabled, Ansible halts execution while the debugger is active, and runs the debugged task immediately after you enter the `redo` command. With the `free` strategy enabled, however, Ansible does not wait for all hosts, and may queue later tasks on one host before a task fails on another host. With the `free` strategy, Ansible does not queue or execute any tasks while the debugger is active. However, all queued tasks remain in the queue and run as soon as you exit the debugger. If you use `redo` to reschedule a task from the debugger, other queued tasks may execute before your rescheduled task. For more information about strategies, see *Controlling playbook execution: strategies and more*.

See also:

Executing playbooks for troubleshooting
Running playbooks while debugging or testing

Ansible playbooks
An introduction to playbooks

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Asynchronous actions and polling

By default Ansible runs tasks synchronously, holding the connection to the remote node open until the action is completed. This means within a playbook, each task blocks the next task by default, meaning subsequent tasks will not run until the current task completes. This behavior can create challenges. For example, a task may take longer to complete than the SSH session allows for, causing a timeout. Or you may want a long-running process to execute in the background while you perform other tasks concurrently. Asynchronous mode lets you control how long-running tasks execute.

Asynchronous ad hoc tasks

You can execute long-running operations in the background with *ad hoc tasks*. For example, to execute `long_running_operation` asynchronously in the background, with a timeout (`-B`) of 3600 seconds, and without polling (`-P`):

```
$ ansible all -B 3600 -P 0 -a "/usr/bin/long_running_operation --do-stuff"
```

To check on the job status later, use the `async_status` module, passing it the job ID that was returned when you ran the original job in the background:

```
$ ansible web1.example.com -m async_status -a "jid=488359678239.2844"
```

Ansible can also check on the status of your long-running job automatically with polling. In most cases, Ansible will keep the connection to your remote node open between polls. To run for 30 minutes and poll for status every 60 seconds:

```
$ ansible all -B 1800 -P 60 -a "/usr/bin/long_running_operation --do-stuff"
```

Poll mode is smart so all jobs will be started before polling begins on any machine. Be sure to use a high enough `--forks` value if you want to get all of your jobs started very quickly. After the time limit (in seconds) runs out (`-B`), the process on the remote nodes will be terminated.

Asynchronous mode is best suited to long-running shell commands or software upgrades. Running the copy module asynchronously, for example, does not do a background file transfer.

Asynchronous playbook tasks

Playbooks also support asynchronous mode and polling, with a simplified syntax. You can use asynchronous mode in playbooks to avoid connection timeouts or to avoid blocking subsequent tasks. The behavior of asynchronous mode in a playbook depends on the value of *poll*.

Avoid connection timeouts: poll > 0

If you want to set a longer timeout limit for a certain task in your playbook, use `async` with `poll` set to a positive value. Ansible will still block the next task in your playbook, waiting until the async task either completes, fails or times out. However, the task will only time out if it exceeds the timeout limit you set with the `async` parameter.

To avoid timeouts on a task, specify its maximum runtime and how frequently you would like to poll for status:

```
---

- hosts: all
  remote_user: root

  tasks:

  - name: Simulate long running op (15 sec), wait for up to 45 sec, poll every 5 sec
    ansible.builtin.command: /bin/sleep 15
    async: 45
    poll: 5
```

Note: The default poll value is set by the *DEFAULT_POLL_INTERVAL* setting. There is no default for the async time limit. If you leave off the 'async' keyword, the task runs synchronously, which is Ansible's default.

Note: As of Ansible 2.3, async does not support check mode and will fail the task when run in check mode. See *Validating tasks: check mode and diff mode* on how to skip a task in check mode.

Note: When an async task completes with polling enabled, the temporary async job cache file (by default in ~/.ansible_async/) is automatically removed.

Run tasks concurrently: poll = 0

If you want to run multiple tasks in a playbook concurrently, use `async` with `poll` set to 0. When you set `poll: 0`, Ansible starts the task and immediately moves on to the next task without waiting for a result. Each async task runs until it either completes, fails or times out (runs longer than its `async` value). The playbook run ends without checking back on async tasks.

To run a playbook task asynchronously:

```
---

- hosts: all
  remote_user: root

  tasks:

  - name: Simulate long running op, allow to run for 45 sec, fire and forget
    ansible.builtin.command: /bin/sleep 15
    async: 45
    poll: 0
```

Note: Do not specify a poll value of 0 with operations that require exclusive locks (such as yum transactions) if you expect to run other commands later in the playbook against those same resources.

Note: Using a higher value for `--forks` will result in kicking off asynchronous tasks even faster. This also increases the efficiency of polling.

Note: When running with `poll: 0`, Ansible will not automatically cleanup the async job cache file. You will need to manually clean this up with the async_status module with `mode: cleanup`.

If you need a synchronization point with an async task, you can register it to obtain its job ID and use the async_status module to observe it in a later task. For example:

```
- name: Run an async task
  ansible.builtin.yum:
    name: docker-io
    state: present
  async: 1000
  poll: 0
  register: yum_sleeper

- name: Check on an async task
  async_status:
    jid: "{{ yum_sleeper.ansible_job_id }}"
  register: job_result
  until: job_result.finished
  retries: 100
  delay: 10
```

Note: If the value of `async:` is not high enough, this will cause the "check on it later" task to fail because the temporary status file that the `async_status:` is looking for will not have been written or no longer exist

Note: Asynchronous playbook tasks always return changed. If the task is using a module that requires the user to annotate changes with `changed_when`, `creates`, and so on, then those should be added to the following `async_status` task.

To run multiple asynchronous tasks while limiting the number of tasks running concurrently:

```
#####################
# main.yml
#####################
- name: Run items asynchronously in batch of two items
  vars:
    sleep_durations:
      - 1
      - 2
      - 3
```

(continues on next page)

(continued from previous page)

```
      - 4
      - 5
    durations: "{{ item }}"
  include_tasks: execute_batch.yml
  loop: "{{ sleep_durations | batch(2) | list }}"

#####################
# execute_batch.yml
#####################
- name: Async sleeping for batched_items
  ansible.builtin.command: sleep {{ async_item }}
  async: 45
  poll: 0
  loop: "{{ durations }}"
  loop_control:
    loop_var: "async_item"
  register: async_results

- name: Check sync status
  async_status:
    jid: "{{ async_result_item.ansible_job_id }}"
  loop: "{{ async_results.results }}"
  loop_control:
    loop_var: "async_result_item"
  register: async_poll_results
  until: async_poll_results.finished
  retries: 30
```

See also:

Controlling playbook execution: strategies and more
    Options for controlling playbook execution

Ansible playbooks
    An introduction to playbooks

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Controlling playbook execution: strategies and more

By default, Ansible runs each task on all hosts affected by a play before starting the next task on any host, using 5 forks. If you want to change this default behavior, you can use a different strategy plugin, change the number of forks, or apply one of several keywords like `serial`.

Selecting a strategy

The default behavior described above is the linear strategy. Ansible offers other strategies, including the debug strategy (see also *Debugging tasks*) and the free strategy, which allows each host to run until the end of the play as fast as it can:

```
- hosts: all
  strategy: free
  tasks:
  # ...
```

You can select a different strategy for each play as shown above, or set your preferred strategy globally in `ansible.cfg`, under the `defaults` stanza:

```
[defaults]
strategy = free
```

All strategies are implemented as *strategy plugins*. Please review the documentation for each strategy plugin for details on how it works.

Setting the number of forks

If you have the processing power available and want to use more forks, you can set the number in `ansible.cfg`:

```
[defaults]
forks = 30
```

or pass it on the command line: *ansible-playbook -f 30 my_playbook.yml*.

Using keywords to control execution

In addition to strategies, several *keywords* also affect play execution. You can set a number, a percentage, or a list of numbers of hosts you want to manage at a time with `serial`. Ansible completes the play on the specified number or percentage of hosts before starting the next batch of hosts. You can restrict the number of workers allotted to a block or task with `throttle`. You can control how Ansible selects the next host in a group to execute against with `order`. You can run a task on a single host with `run_once`. These keywords are not strategies. They are directives or options applied to a play, block, or task.

Other keywords that affect play execution include `ignore_errors`, `ignore_unreachable`, and `any_errors_fatal`. These options are documented in *Error handling in playbooks*.

Setting the batch size with `serial`

By default, Ansible runs in parallel against all the hosts in the *pattern* you set in the `hosts:` field of each play. If you want to manage only a few machines at a time, for example during a rolling update, you can define how many hosts Ansible should manage at a single time using the `serial` keyword:

```
---
- name: test play
  hosts: webservers
  serial: 3
  gather_facts: False

  tasks:
    - name: first task
      command: hostname
    - name: second task
      command: hostname
```

In the above example, if we had 6 hosts in the group 'webservers', Ansible would execute the play completely (both tasks) on 3 of the hosts before moving on to the next 3 hosts:

```
PLAY [webservers] ****************************************

TASK [first task] ****************************************
changed: [web3]
changed: [web2]
changed: [web1]

TASK [second task] ***************************************
changed: [web1]
changed: [web2]
changed: [web3]

PLAY [webservers] ****************************************

TASK [first task] ****************************************
changed: [web4]
changed: [web5]
changed: [web6]

TASK [second task] ***************************************
changed: [web4]
changed: [web5]
changed: [web6]

PLAY RECAP ***********************************************
web1      : ok=2    changed=2    unreachable=0    failed=0
web2      : ok=2    changed=2    unreachable=0    failed=0
web3      : ok=2    changed=2    unreachable=0    failed=0
web4      : ok=2    changed=2    unreachable=0    failed=0
web5      : ok=2    changed=2    unreachable=0    failed=0
web6      : ok=2    changed=2    unreachable=0    failed=0
```

Note: Setting the batch size with `serial` changes the scope of the Ansible failures to the batch size, not the entire host list. You can use *ignore_unreachable* or *max_fail_percentage* to modify this behavior.

You can also specify a percentage with the `serial` keyword. Ansible applies the percentage to the total number of hosts in a play to determine the number of hosts per pass:

```
---
- name: test play
  hosts: webservers
  serial: "30%"
```

If the number of hosts does not divide equally into the number of passes, the final pass contains the remainder. In this example, if you had 20 hosts in the webservers group, the first batch would contain 6 hosts, the second batch would contain 6 hosts, the third batch would contain 6 hosts, and the last batch would contain 2 hosts.

You can also specify batch sizes as a list. For example:

```
---
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - 10
```

In the above example, the first batch would contain a single host, the next would contain 5 hosts, and (if there are any hosts left), every following batch would contain either 10 hosts or all the remaining hosts, if fewer than 10 hosts remained.

You can list multiple batch sizes as percentages:

```
---
- name: test play
  hosts: webservers
  serial:
    - "10%"
    - "20%"
    - "100%"
```

You can also mix and match the values:

```
---
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - "20%"
```

Note: No matter how small the percentage, the number of hosts per pass will always be 1 or greater.

Restricting execution with throttle

The throttle keyword limits the number of workers for a particular task. It can be set at the block and task level. Use throttle to restrict tasks that may be CPU-intensive or interact with a rate-limiting API:

```
tasks:
- command: /path/to/cpu_intensive_command
  throttle: 1
```

If you have already restricted the number of forks or the number of machines to execute against in parallel, you can reduce the number of workers with throttle, but you cannot increase it. In other words, to have an effect, your throttle setting must be lower than your forks or serial setting if you are using them together.

Ordering execution based on inventory

The order keyword controls the order in which hosts are run. Possible values for order are:

inventory:
(default) The order provided by the inventory for the selection requested (see note below)

reverse_inventory:
The same as above, but reversing the returned list

sorted:
Sorted alphabetically sorted by name

reverse_sorted:
Sorted by name in reverse alphabetical order

shuffle:
Randomly ordered on each run

Note: the 'inventory' order does not equate to the order in which hosts/groups are defined in the inventory source file, but the 'order in which a selection is returned from the compiled inventory'. This is a backwards compatible option and while reproducible it is not normally predictable. Due to the nature of inventory, host patterns, limits, inventory plugins and the ability to allow multiple sources it is almost impossible to return such an order. For simple cases this might happen to match the file definition order, but that is not guaranteed.

Running on a single machine with run_once

If you want a task to run only on the first host in your batch of hosts, set run_once to true on that task:

```
---
# ...

  tasks:

    # ...

    - command: /opt/application/upgrade_db.py
      run_once: true

    # ...
```

Ansible executes this task on the first host in the current batch and applies all results and facts to all the hosts in the same batch. This approach is similar to applying a conditional to a task such as:

```
- command: /opt/application/upgrade_db.py
  when: inventory_hostname == webservers[0]
```

However, with `run_once`, the results are applied to all the hosts. To run the task on a specific host, instead of the first host in the batch, delegate the task:

```
- command: /opt/application/upgrade_db.py
  run_once: true
  delegate_to: web01.example.org
```

As always with *delegation*, the action will be executed on the delegated host, but the information is still that of the original host in the task.

Note: When used together with `serial`, tasks marked as `run_once` will be run on one host in *each* serial batch. If the task must run only once regardless of `serial` mode, use `when: inventory_hostname == ansible_play_hosts_all[0]` construct.

Note: Any conditional (in other words, *when:*) will use the variables of the 'first host' to decide if the task runs or not, no other hosts will be tested.

Note: If you want to avoid the default behavior of setting the fact for all hosts, set `delegate_facts: True` for the specific task or block.

See also:

Ansible playbooks
An introduction to playbooks

Controlling where tasks run: delegation and local actions
Running tasks on or assigning facts to specific machines

Roles
Playbook organization by roles

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

1.6.4 Advanced playbook syntax

The advanced YAML syntax examples on this page give you more control over the data placed in YAML files used by Ansible. You can find additional information about Python-specific YAML in the official PyYAML Documentation.

Unsafe or raw strings

When handling values returned by lookup plugins, Ansible uses a data type called `unsafe` to block templating. Marking data as unsafe prevents malicious users from abusing Jinja2 templates to execute arbitrary code on target machines. The Ansible implementation ensures that unsafe values are never templated. It is more comprehensive than escaping Jinja2 with `{% raw %} ... {% endraw %}` tags.

You can use the same `unsafe` data type in variables you define, to prevent templating errors and information disclosure. You can mark values supplied by *vars_prompts* as unsafe. You can also use `unsafe` in playbooks. The most common use cases include passwords that allow special characters like `{` or `%`, and JSON arguments that look like templates but should not be templated. For example:

```
---
mypassword: !unsafe 234%234{435lkj{{lkjsdf
```

In a playbook:

```
---
hosts: all
vars:
    my_unsafe_variable: !unsafe 'unsafe % value'
tasks:
    ...
```

For complex variables such as hashes or arrays, use `!unsafe` on the individual elements:

```
---
my_unsafe_array:
    - !unsafe 'unsafe element'
    - 'safe element'

my_unsafe_hash:
    unsafe_key: !unsafe 'unsafe value'
```

YAML anchors and aliases: sharing variable values

YAML anchors and aliases help you define, maintain, and use shared variable values in a flexible way. You define an anchor with `&`, then refer to it using an alias, denoted with `*`. Here's an example that sets three values with an anchor, uses two of those values with an alias, and overrides the third value:

```
---
...
vars:
    app1:
        jvm: &jvm_opts
            opts: '-Xms1G -Xmx2G'
            port: 1000
            path: /usr/lib/app1
```

(continues on next page)

(continued from previous page)

```
    app2:
        jvm:
            <<: *jvm_opts
            path: /usr/lib/app2
...
```

Here, app1 and app2 share the values for opts and port using the anchor &jvm_opts and the alias *jvm_opts. The value for path is merged by << or merge operator.

Anchors and aliases also let you share complex sets of variable values, including nested variables. If you have one variable value that includes another variable value, you can define them separately:

```
vars:
  webapp_version: 1.0
  webapp_custom_name: ToDo_App-1.0
```

This is inefficient and, at scale, means more maintenance. To incorporate the version value in the name, you can use an anchor in app_version and an alias in custom_name:

```
vars:
  webapp:
      version: &my_version 1.0
      custom_name:
          - "ToDo_App"
          - *my_version
```

Now, you can re-use the value of app_version within the value of custom_name and use the output in a template:

```
---
- name: Using values nested inside dictionary
  hosts: localhost
  vars:
    webapp:
        version: &my_version 1.0
        custom_name:
            - "ToDo_App"
            - *my_version
  tasks:
  - name: Using Anchor value
    ansible.builtin.debug:
        msg: My app is called "{{ webapp.custom_name | join('-') }}".
```

You've anchored the value of version with the &my_version anchor, and re-used it with the *my_version alias. Anchors and aliases let you access nested values inside dictionaries.

See also:

Using Variables
    All about variables

Manipulating data
    Doing complex data manipulation in Ansible

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

1.6.5 Manipulating data

In many cases, you need to do some complex operation with your variables, while Ansible is not recommended as a data processing/manipulation tool, you can use the existing Jinja2 templating in conjunction with the many added Ansible filters, lookups and tests to do some very complex transformations.

Let's start with a quick definition of each type of plugin:

- lookups: Mainly used to query 'external data', in Ansible these were the primary part of loops using the `with_<lookup>` construct, but they can be used independently to return data for processing. They normally return a list due to their primary function in loops as mentioned previously. Used with the `lookup` or `query` Jinja2 operators.
- filters: used to change/transform data, used with the `|` Jinja2 operator.
- tests: used to validate data, used with the `is` Jinja2 operator.

Loops and list comprehensions

Most programming languages have loops (`for`, `while`, and so on) and list comprehensions to do transformations on lists including lists of objects. Jinja2 has a few filters that provide this functionality: `map`, `select`, `reject`, `selectattr`, `rejectattr`.

- map: this is a basic for loop that just allows you to change every item in a list, using the 'attribute' keyword you can do the transformation based on attributes of the list elements.
- select/reject: this is a for loop with a condition, that allows you to create a subset of a list that matches (or not) based on the result of the condition.
- selectattr/rejectattr: very similar to the above but it uses a specific attribute of the list elements for the conditional statement.

Use a loop to create exponential backoff for retries/until.

```
- name: retry ping 10 times with exponential backoff delay
  ping:
  retries: 10
  delay: '{{item|int}}'
  loop: '{{ range(1, 10)|map("pow", 2) }}'
```

Extract keys from a dictionary matching elements from a list

The Python equivalent code would be:

```
chains = [1, 2]
for chain in chains:
    for config in chains_config[chain]['configs']:
        print(config['type'])
```

There are several ways to do it in Ansible, this is just one example:

Listing 6: Way to extract matching keys from a list of dictionaries

```
tasks:
  - name: Show extracted list of keys from a list of dictionaries
    ansible.builtin.debug:
      msg: "{{ chains | map('extract', chains_config) | map(attribute='configs') |␣
↪flatten | map(attribute='type') | flatten }}"
    vars:
      chains: [1, 2]
      chains_config:
          1:
              foo: bar
              configs:
                  - type: routed
                    version: 0.1
                  - type: bridged
                    version: 0.2
          2:
              foo: baz
              configs:
                  - type: routed
                    version: 1.0
                  - type: bridged
                    version: 1.1
```

Listing 7: Results of debug task, a list with the extracted keys

```
  ok: [localhost] => {
      "msg": [
          "routed",
          "bridged",
          "routed",
          "bridged"
      ]
  }
```

Listing 8: Get the unique list of values of a variable that vary per host

```
  vars:
      unique_value_list: "{{ groups['all'] | map ('extract', hostvars, 'varname') |␣
↪list | unique}}"
```

Find mount point

In this case, we want to find the mount point for a given path across our machines, since we already collect mount facts, we can use the following:

Listing 9: Use selectattr to filter mounts into list I can then sort and select the last from

```
- hosts: all
  gather_facts: True
  vars:
     path: /var/lib/cache
  tasks:
  - name: The mount point for {{path}}, found using the Ansible mount facts, [-1] is␣
↪the same as the 'last' filter
    ansible.builtin.debug:
     msg: "{{(ansible_facts.mounts | selectattr('mount', 'in', path) | list |␣
↪sort(attribute='mount'))[-1]['mount']}}"
```

Omit elements from a list

The special omit variable ONLY works with module options, but we can still use it in other ways as an identifier to tailor a list of elements:

Listing 10: Inline list filtering when feeding a module option

```
- name: Enable a list of Windows features, by name
  ansible.builtin.set_fact:
    win_feature_list: "{{ namestuff | reject('equalto', omit) | list }}"
  vars:
    namestuff:
      - "{{ (fs_installed_smb_v1 | default(False)) | ternary(omit, 'FS-SMB1') }}"
      - "foo"
      - "bar"
```

Another way is to avoid adding elements to the list in the first place, so you can just use it directly:

Listing 11: Using set_fact in a loop to increment a list conditionally

```
- name: Build unique list with some items conditionally omitted
  ansible.builtin.set_fact:
     namestuff: ' {{ (namestuff | default([])) | union([item]) }}'
  when: item != omit
  loop:
      - "{{ (fs_installed_smb_v1 | default(False)) | ternary(omit, 'FS-SMB1') }}"
      - "foo"
      - "bar"
```

Combine values from same list of dicts

Combining positive and negative filters from examples above, you can get a 'value when it exists' and a 'fallback' when it doesn't.

Listing 12: Use selectattr and rejectattr to get the ansible_host or inventory_hostname as needed

```
- hosts: localhost
  tasks:
    - name: Check hosts in inventory that respond to ssh port
      wait_for:
        host: "{{ item }}"
        port: 22
      loop: '{{ has_ah + no_ah }}'
      vars:
        has_ah: '{{ hostvars|dictsort|selectattr("1.ansible_host", "defined
↪")|map(attribute="1.ansible_host")|list }}'
        no_ah: '{{ hostvars|dictsort|rejectattr("1.ansible_host", "defined
↪")|map(attribute="0")|list }}'
```

Custom Fileglob Based on a Variable

This example uses Python argument list unpacking to create a custom list of fileglobs based on a variable.

Listing 13: Using fileglob with a list based on a variable.

```
- hosts: all
  vars:
    mygroups:
      - prod
      - web
  tasks:
    - name: Copy a glob of files based on a list of groups
      copy:
        src: "{{ item }}"
        dest: "/tmp/{{ item }}"
      loop: '{{ q("fileglob", *globlist) }}'
      vars:
```

(continues on next page)

(continued from previous page)

```
        globlist: '{{ mygroups | map("regex_replace", "^(.*)$", "files/\1/*.conf") |␣
↪list }}'
```

Complex Type transformations

Jinja provides filters for simple data type transformations (`int`, `bool`, and so on), but when you want to transform data structures things are not as easy. You can use loops and list comprehensions as shown above to help, also other filters and lookups can be chained and used to achieve more complex transformations.

Create dictionary from list

In most languages it is easy to create a dictionary (a.k.a. map/associative array/hash and so on) from a list of pairs, in Ansible there are a couple of ways to do it and the best one for you might depend on the source of your data.

These example produces `{"a": "b", "c": "d"}`

Listing 14: Simple list to dict by assuming the list is [key, value , key, value, …]

```
vars:
    single_list: [ 'a', 'b', 'c', 'd' ]
    mydict: "{{ dict(single_list[::2] | zip_longest(single_list[1::2])) }}"
```

Listing 15: It is simpler when we have a list of pairs:

```
vars:
    list_of_pairs: [ ['a', 'b'], ['c', 'd'] ]
    mydict: "{{ dict(list_of_pairs) }}"
```

Both end up being the same thing, with `zip_longest` transforming `single_list` to a `list_of_pairs` generator.

A bit more complex, using `set_fact` and a `loop` to create/update a dictionary with key value pairs from 2 lists:

Listing 16: Using set_fact to create a dictionary from a set of lists

```
  - name: Uses 'combine' to update the dictionary and 'zip' to make pairs of both lists
    ansible.builtin.set_fact:
      mydict: "{{ mydict | default({}) | combine({item[0]: item[1]}) }}"
    loop: "{{ (keys | zip(values)) | list }}"
    vars:
      keys:
        - foo
        - var
        - bar
      values:
        - a
        - b
        - c
```

This results in `{"foo": "a", "var": "b", "bar": "c"}`.

You can even combine these simple examples with other filters and lookups to create a dictionary dynamically by matching patterns to variable names:

Listing 17: Using 'vars' to define dictionary from a set of lists without needing a task

```
  vars:
      xyz_stuff: 1234
      xyz_morestuff: 567
      myvarnames: "{{ q('varnames', '^xyz_') }}"
      mydict: "{{ dict(myvarnames|map('regex_replace', '^xyz_', '')|list | zip(q('vars',
↪ *myvarnames))) }}"
```

A quick explanation, since there is a lot to unpack from these two lines:

- The `varnames` lookup returns a list of variables that match "begin with `xyz_`".
- Then feeding the list from the previous step into the `vars` lookup to get the list of values. The `*` is used to 'dereference the list' (a pythonism that works in Jinja), otherwise it would take the list as a single argument.
- Both lists get passed to the `zip` filter to pair them off into a unified list (key, value, key2, value2, …).
- The dict function then takes this 'list of pairs' to create the dictionary.

An example on how to use facts to find a host's data that meets condition X:

```
vars:
  uptime_of_host_most_recently_rebooted: "{{ansible_play_hosts_all | map('extract',␣
↪hostvars, 'ansible_uptime_seconds') | sort | first}}"
```

An example to show a host uptime in days/hours/minutes/seconds (assumes facts were gathered).

```
- name: Show the uptime in days/hours/minutes/seconds
  ansible.builtin.debug:
   msg: Uptime {{ now().replace(microsecond=0) - now().fromtimestamp(now(fmt='%s') | int␣
↪- ansible_uptime_seconds) }}
```

See also:

Using filters to manipulate data
Jinja2 filters included with Ansible

Tests
Jinja2 tests included with Ansible

Jinja2 Docs
Jinja2 documentation, includes lists for core filters and tests

1.7 Protecting sensitive data with Ansible vault

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible vault documentation. Ansible vault provides a way to encrypt and manage sensitive data such as passwords. This guide introduces you to Ansible vault and covers the following topics:

- Managing vault passwords.
- Encrypting content and files with Ansible vault.
- Using encrypted variables and files.

1.7.1 Ansible Vault

Ansible Vault encrypts variables and files so you can protect sensitive content such as passwords or keys rather than leaving it visible as plaintext in playbooks or roles. To use Ansible Vault you need one or more passwords to encrypt and decrypt content. If you store your vault passwords in a third-party tool such as a secret manager, you need a script to access them. Use the passwords with the *ansible-vault* command-line tool to create and view encrypted variables, create encrypted files, encrypt existing files, or edit, re-key, or decrypt files. You can then place encrypted content under source control and share it more safely.

Warning:

- Encryption with Ansible Vault ONLY protects 'data at rest'. Once the content is decrypted ('data in use'), play and plugin authors are responsible for avoiding any secret disclosure, see *no_log* for details on hiding output and *Steps to secure your editor* for security considerations on editors you use with Ansible Vault.

You can use encrypted variables and files in ad hoc commands and playbooks by supplying the passwords you used to encrypt them. You can modify your `ansible.cfg` file to specify the location of a password file or to always prompt for the password.

1.7.2 Managing vault passwords

Managing your encrypted content is easier if you develop a strategy for managing your vault passwords. A vault password can be any string you choose. There is no special command to create a vault password. However, you need to keep track of your vault passwords. Each time you encrypt a variable or file with Ansible Vault, you must provide a password. When you use an encrypted variable or file in a command or playbook, you must provide the same password that was used to encrypt it. To develop a strategy for managing vault passwords, start with two questions:

- Do you want to encrypt all your content with the same password, or use different passwords for different needs?
- Where do you want to store your password or passwords?

Choosing between a single password and multiple passwords

If you have a small team or few sensitive values, you can use a single password for everything you encrypt with Ansible Vault. Store your vault password securely in a file or a secret manager as described below.

If you have a larger team or many sensitive values, you can use multiple passwords. For example, you can use different passwords for different users or different levels of access. Depending on your needs, you might want a different password for each encrypted file, for each directory, or for each environment. For example, you might have a playbook that includes two vars files, one for the dev environment and one for the production environment, encrypted with two different passwords. When you run the playbook, select the correct vault password for the environment you are targeting, using a vault ID.

Managing multiple passwords with vault IDs

If you use multiple vault passwords, you can differentiate one password from another with vault IDs. You use the vault ID in three ways:

- Pass it with *--vault-id* to the *ansible-vault* command when you create encrypted content
- Include it wherever you store the password for that vault ID (see *Storing and accessing vault passwords*)
- Pass it with *--vault-id* to the *ansible-playbook* command when you run a playbook that uses content you encrypted with that vault ID

When you pass a vault ID as an option to the *ansible-vault* command, you add a label (a hint or nickname) to the encrypted content. This label documents which password you used to encrypt it. The encrypted variable or file includes the vault ID label in plain text in the header. The vault ID is the last element before the encrypted content. For example:

```
my_encrypted_var: !vault |
          $ANSIBLE_VAULT;1.2;AES256;dev
          ␣
↪30613233633461343837653833666333643061636561303338373661313838333565653635353162
          ␣
↪3263363434623733343538653462613064333634333464660a663633623939393439316636633863
          ␣
↪61636237636537333938306331383339353265363239643939666639386530626330633337633833
          ␣
↪6664656334373166630a363736393262666465663432613932613036303963343263623137386239
          6330
```

In addition to the label, you must provide a source for the related password. The source can be a prompt, a file, or a script, depending on how you are storing your vault passwords. The pattern looks like this:

```
--vault-id label@source
```

If your playbook uses multiple encrypted variables or files that you encrypted with different passwords, you must pass the vault IDs when you run that playbook. You can use *--vault-id* by itself, with *--vault-password-file*, or with *--ask-vault-pass*. The pattern is the same as when you create encrypted content: include the label and the source for the matching password.

See below for examples of encrypting content with vault IDs and using content encrypted with vault IDs. The *--vault-id* option works with any Ansible command that interacts with vaults, including *ansible-vault*, *ansible-playbook*, and so on.

Limitations of vault IDs

Ansible does not enforce using the same password every time you use a particular vault ID label. You can encrypt different variables or files with the same vault ID label but different passwords. This usually happens when you type the password at a prompt and make a mistake. It is possible to use different passwords with the same vault ID label on purpose. For example, you could use each label as a reference to a class of passwords, rather than a single password. In this scenario, you must always know which specific password or file to use in context. However, you are more likely to encrypt two files with the same vault ID label and different passwords by mistake. If you encrypt two files with the same label but different passwords by accident, you can *rekey* one file to fix the issue.

Enforcing vault ID matching

By default the vault ID label is only a hint to remind you which password you used to encrypt a variable or file. Ansible does not check that the vault ID in the header of the encrypted content matches the vault ID you provide when you use the content. Ansible decrypts all files and variables called by your command or playbook that are encrypted with the password you provide. To check the encrypted content and decrypt it only when the vault ID it contains matches the one you provide with `--vault-id`, set the config option *DEFAULT_VAULT_ID_MATCH*. When you set *DEFAULT_VAULT_ID_MATCH*, each password is only used to decrypt data that was encrypted with the same label. This is efficient, predictable, and can reduce errors when different values are encrypted with different passwords.

Note: Even with the *DEFAULT_VAULT_ID_MATCH* setting enabled, Ansible does not enforce using the same password every time you use a particular vault ID label.

Storing and accessing vault passwords

You can memorize your vault password, or manually copy vault passwords from any source and paste them at a command-line prompt, but most users store them securely and access them as needed from within Ansible. You have two options for storing vault passwords that work from within Ansible: in files, or in a third-party tool such as the system keyring or a secret manager. If you store your passwords in a third-party tool, you need a vault password client script to retrieve them from within Ansible.

Storing passwords in files

To store a vault password in a file, enter the password as a string on a single line in the file. Make sure the permissions on the file are appropriate. Do not add password files to source control.

Storing passwords in third-party tools with vault password client scripts

You can store your vault passwords on the system keyring, in a database, or in a secret manager and retrieve them from within Ansible using a vault password client script. Enter the password as a string on a single line. If your password has a vault ID, store it in a way that works with your password storage tool.

To create a vault password client script:

- Create a file with a name ending in either `-client` or `-client.EXTENSION`
- Make the file executable
- **Within the script itself:**
  - Print the passwords to standard output
  - Accept a `--vault-id` option
  - If the script prompts for data (for example, a database password), display the prompts to the TTY.

When you run a playbook that uses vault passwords stored in a third-party tool, specify the script as the source within the `--vault-id` flag. For example:

```
ansible-playbook --vault-id dev@contrib/vault/vault-keyring-client.py
```

Ansible executes the client script with a `--vault-id` option so the script knows which vault ID label you specified. For example a script loading passwords from a secret manager can use the vault ID label to pick either the 'dev' or 'prod' password. The example command above results in the following execution of the client script:

```
contrib/vault/vault-keyring-client.py --vault-id dev
```

For an example of a client script that loads passwords from the system keyring, see the vault-keyring-client script.

1.7.3 Encrypting content with Ansible Vault

Once you have a strategy for managing and storing vault passwords, you can start encrypting content. You can encrypt two types of content with Ansible Vault: variables and files. Encrypted content always includes the `!vault` tag, which tells Ansible and YAML that the content needs to be decrypted, and a `|` character, which allows multi-line strings. Encrypted content created with `--vault-id` also contains the vault ID label. For more details about the encryption process and the format of content encrypted with Ansible Vault, see *Format of files encrypted with Ansible Vault*. This table shows the main differences between encrypted variables and encrypted files:

	Encrypted variables	Encrypted files
How much is encrypted?	Variables within a plaintext file	The entire file
When is it decrypted?	On demand, only when needed	Whenever loaded or referenced[1]
What can be encrypted?	Only variables	Any structured data file

Encrypting individual variables with Ansible Vault

You can encrypt single values inside a YAML file using the *ansible-vault encrypt_string* command. For one way to keep your vaulted variables safely visible, see *Keep vaulted variables safely visible*.

Advantages and disadvantages of encrypting variables

With variable-level encryption, your files are still easily legible. You can mix plaintext and encrypted variables, even inline in a play or role. However, password rotation is not as simple as with file-level encryption. You cannot *rekey* encrypted variables. Also, variable-level encryption only works on variables. If you want to encrypt tasks or other content, you must encrypt the entire file.

Creating encrypted variables

The *ansible-vault encrypt_string* command encrypts and formats any string you type (or copy or generate) into a format that can be included in a playbook, role, or variables file. To create a basic encrypted variable, pass three options to the *ansible-vault encrypt_string* command:

- a source for the vault password (prompt, file, or script, with or without a vault ID)
- the string to encrypt
- the string name (the name of the variable)

The pattern looks like this:

```
ansible-vault encrypt_string <password_source> '<string_to_encrypt>' --name '<string_
↪name_of_variable>'
```

[1] Ansible cannot know if it needs content from an encrypted file unless it decrypts the file, so it decrypts all encrypted files referenced in your playbooks and roles.

For example, to encrypt the string ‘foobar’ using the only password stored in ‘a_password_file’ and name the variable ‘the_secret’:

```
ansible-vault encrypt_string --vault-password-file a_password_file 'foobar' --name 'the_
↪secret'
```

The command above creates this content:

```
the_secret: !vault |
      $ANSIBLE_VAULT;1.1;AES256
     ␣
↪62313365396662343061393464336163383764373764613633653634306231386433626436623361
     ␣
↪6134333665353966363534333632666535333761666131620a663537646436643839616531643561
     ␣
↪63396265333966386166373632626539326166353965363262633030333630313338646335303630
     ␣
↪3438626666666137650a353638643435666633633964366338633066623234616432373231333331
      6564
```

To encrypt the string ‘foooodev’, add the vault ID label ‘dev’ with the ‘dev’ vault password stored in ‘a_password_file’, and call the encrypted variable ‘the_dev_secret’:

```
ansible-vault encrypt_string --vault-id dev@a_password_file 'foooodev' --name 'the_dev_
↪secret'
```

The command above creates this content:

```
the_dev_secret: !vault |
          $ANSIBLE_VAULT;1.2;AES256;dev
         ␣
↪30613233633461343837653833666333643061636561303338373661313838333565653635353162
         ␣
↪3263363434623733343538653462613064333634333464660a663633623939393439316636633863
         ␣
↪61636237636537333938306331383339353265363239643939666639386530626330633337633833
         ␣
↪6664656334373166630a363736393262666465663432613932613036303963343263623137386239
          6330
```

To encrypt the string ‘letmein’ read from stdin, add the vault ID ‘dev’ using the ‘dev’ vault password stored in *a_password_file*, and name the variable ‘db_password’:

```
echo -n 'letmein' | ansible-vault encrypt_string --vault-id dev@a_password_file --stdin-
↪name 'db_password'
```

Warning: Typing secret content directly at the command line (without a prompt) leaves the secret string in your shell history. Do not do this outside of testing.

The command above creates this output:

```
Reading plaintext input from stdin. (ctrl-d to end input, twice if your␣
↪content does not already have a new line)
```

(continues on next page)

(continued from previous page)

```
db_password: !vault |
          $ANSIBLE_VAULT;1.2;AES256;dev
          61323931353866666336306139373937316366366138656131323863373866376666353364373761
          35396332343138363464353237663061646261343765643330a37353031363534353534313331613
          36643666306434616266376434363239346433643238336464643566386135356334303736353136
          6565633133366366360a326566323363363936613664616364623437336130623133343530333739
          3039
```

To be prompted for a string to encrypt, encrypt it with the ‘dev’ vault password from ‘a_password_file’, name the variable ‘new_user_password’ and give it the vault ID label ‘dev’:

```
ansible-vault encrypt_string --vault-id dev@a_password_file --stdin-name 'new_user_password'
```

The command above triggers this prompt:

```
Reading plaintext input from stdin. (ctrl-d to end input, twice if your content does not already have a new line)
```

Type the string to encrypt (for example, ‘hunter2’), hit ctrl-d, and wait.

Warning: Do not press `Enter` after supplying the string to encrypt. That will add a newline to the encrypted value.

The sequence above creates this output:

```
new_user_password: !vault |
          $ANSIBLE_VAULT;1.2;AES256;dev
          37636561366636643464376336303466613062633537323632306566653533383833366462366662
          6565353063303065303831323539656138653863353230620a653638643639333133306331336365
          62373737623337616130386137373461306535383538373162316263386165376131623631323434
          3866363862363335620a376466656164383032633338306162326639643635663936623939666238
          3161
```

You can add the output from any of the examples above to any playbook, variables file, or role for future use. Encrypted variables are larger than plain-text variables, but they protect your sensitive content while leaving the rest of the playbook, variables file, or role in plain text so you can easily read it.

Viewing encrypted variables

You can view the original value of an encrypted variable using the debug module. You must pass the password that was used to encrypt the variable. For example, if you stored the variable created by the last example above in a file called 'vars.yml', you could view the unencrypted value of that variable like this:

```
ansible localhost -m ansible.builtin.debug -a var="new_user_password" -e "@vars.yml" --
↪vault-id dev@a_password_file

localhost | SUCCESS => {
    "new_user_password": "hunter2"
}
```

Encrypting files with Ansible Vault

Ansible Vault can encrypt any structured data file used by Ansible, including:

- group variables files from inventory
- host variables files from inventory
- variables files passed to ansible-playbook with `-e @file.yml` or `-e @file.json`
- variables files loaded by `include_vars` or `vars_files`
- variables files in roles
- defaults files in roles
- tasks files
- handlers files
- binary files or other arbitrary files

The full file is encrypted in the vault.

Note: Ansible Vault uses an editor to create or modify encrypted files. See *Steps to secure your editor* for some guidance on securing the editor.

Advantages and disadvantages of encrypting files

File-level encryption is easy to use. Password rotation for encrypted files is straightforward with the *rekey* command. Encrypting files can hide not only sensitive values, but the names of the variables you use. However, with file-level encryption the contents of files are no longer easy to access and read. This may be a problem with encrypted tasks files. When encrypting a variables file, see *Keep vaulted variables safely visible* for one way to keep references to these variables in a non-encrypted file. Ansible always decrypts the entire encrypted file when it is when loaded or referenced, because Ansible cannot know if it needs the content unless it decrypts it.

Creating encrypted files

To create a new encrypted data file called 'foo.yml' with the 'test' vault password from 'multi_password_file':

```
ansible-vault create --vault-id test@multi_password_file foo.yml
```

The tool launches an editor (whatever editor you have defined with $EDITOR, default editor is vi). Add the content. When you close the editor session, the file is saved as encrypted data. The file header reflects the vault ID used to create it:

```
``$ANSIBLE_VAULT;1.2;AES256;test``
```

To create a new encrypted data file with the vault ID 'my_new_password' assigned to it and be prompted for the password:

```
ansible-vault create --vault-id my_new_password@prompt foo.yml
```

Again, add content to the file in the editor and save. Be sure to store the new password you created at the prompt, so you can find it when you want to decrypt that file.

Encrypting existing files

To encrypt an existing file, use the *ansible-vault encrypt* command. This command can operate on multiple files at once. For example:

```
ansible-vault encrypt foo.yml bar.yml baz.yml
```

To encrypt existing files with the 'project' ID and be prompted for the password:

```
ansible-vault encrypt --vault-id project@prompt foo.yml bar.yml baz.yml
```

Viewing encrypted files

To view the contents of an encrypted file without editing it, you can use the *ansible-vault view* command:

```
ansible-vault view foo.yml bar.yml baz.yml
```

Editing encrypted files

To edit an encrypted file in place, use the *ansible-vault edit* command. This command decrypts the file to a temporary file, allows you to edit the content, then saves and re-encrypts the content and removes the temporary file when you close the editor. For example:

```
ansible-vault edit foo.yml
```

To edit a file encrypted with the `vault2` password file and assigned the vault ID `pass2`:

```
ansible-vault edit --vault-id pass2@vault2 foo.yml
```

Changing the password and/or vault ID on encrypted files

To change the password on an encrypted file or files, use the *rekey* command:

```
ansible-vault rekey foo.yml bar.yml baz.yml
```

This command can rekey multiple data files at once and will ask for the original password and also the new password. To set a different ID for the rekeyed files, pass the new ID to `--new-vault-id`. For example, to rekey a list of files encrypted with the 'preprod1' vault ID from the 'ppold' file to the 'preprod2' vault ID and be prompted for the new password:

```
ansible-vault rekey --vault-id preprod1@ppold --new-vault-id preprod2@prompt foo.yml bar.
↪yml baz.yml
```

Decrypting encrypted files

If you have an encrypted file that you no longer want to keep encrypted, you can permanently decrypt it by running the *ansible-vault decrypt* command. This command will save the file unencrypted to the disk, so be sure you do not want to *edit* it instead.

```
ansible-vault decrypt foo.yml bar.yml baz.yml
```

Steps to secure your editor

Ansible Vault relies on your configured editor, which can be a source of disclosures. Most editors have ways to prevent loss of data, but these normally rely on extra plain text files that can have a clear text copy of your secrets. Consult your editor documentation to configure the editor to avoid disclosing secure data. The following sections provide some guidance on common editors but should not be taken as a complete guide to securing your editor.

vim

You can set the following `vim` options in command mode to avoid cases of disclosure. There may be more settings you need to modify to ensure security, especially when using plugins, so consult the `vim` documentation.

1. Disable swapfiles that act like an autosave in case of crash or interruption.

```
set noswapfile
```

2. Disable creation of backup files.

```
set nobackup
set nowritebackup
```

3. Disable the viminfo file from copying data from your current session.

```
set viminfo=
```

4. Disable copying to the system clipboard.

```
set clipboard=
```

You can optionally add these settings in `.vimrc` for all files, or just specific paths or extensions. See the `vim` manual for details.

Emacs

You can set the following Emacs options to avoid cases of disclosure. There may be more settings you need to modify to ensure security, especially when using plugins, so consult the Emacs documentation.

1. Do not copy data to the system clipboard.

```
(setq x-select-enable-clipboard nil)
```

2. Disable creation of backup files.

```
(setq make-backup-files nil)
```

3. Disable autosave files.

```
(setq auto-save-default nil)
```

1.7.4 Using encrypted variables and files

When you run a task or playbook that uses encrypted variables or files, you must provide the passwords to decrypt the variables or files. You can do this at the command line or by setting a default password source in a config option or an environment variable.

Passing a single password

If all the encrypted variables and files your task or playbook needs use a single password, you can use the *`--ask-vault-pass`* or *`--vault-password-file`* cli options.

To prompt for the password:

```
ansible-playbook --ask-vault-pass site.yml
```

To retrieve the password from the `/path/to/my/vault-password-file` file:

```
ansible-playbook --vault-password-file /path/to/my/vault-password-file site.yml
```

To get the password from the vault password client script `my-vault-password-client.py`:

```
ansible-playbook --vault-password-file my-vault-password-client.py
```

Passing vault IDs

You can also use the `--vault-id` option to pass a single password with its vault label. This approach is clearer when multiple vaults are used within a single inventory.

To prompt for the password for the 'dev' vault ID:

```
ansible-playbook --vault-id dev@prompt site.yml
```

To retrieve the password for the 'dev' vault ID from the `dev-password` file:

```
ansible-playbook --vault-id dev@dev-password site.yml
```

To get the password for the 'dev' vault ID from the vault password client script `my-vault-password-client.py`:

```
ansible-playbook --vault-id dev@my-vault-password-client.py
```

Passing multiple vault passwords

If your task or playbook requires multiple encrypted variables or files that you encrypted with different vault IDs, you must use the `--vault-id` option, passing multiple `--vault-id` options to specify the vault IDs ('dev', 'prod', 'cloud', 'db') and sources for the passwords (prompt, file, script). . For example, to use a 'dev' password read from a file and to be prompted for the 'prod' password:

```
ansible-playbook --vault-id dev@dev-password --vault-id prod@prompt site.yml
```

By default the vault ID labels (dev, prod and so on) are only hints. Ansible attempts to decrypt vault content with each password. The password with the same label as the encrypted data will be tried first, after that each vault secret will be tried in the order they were provided on the command line.

Where the encrypted data has no label, or the label does not match any of the provided labels, the passwords will be tried in the order they are specified. In the example above, the 'dev' password will be tried first, then the 'prod' password for cases where Ansible doesn't know which vault ID is used to encrypt something.

Using `--vault-id` without a vault ID

The `--vault-id` option can also be used without specifying a vault-id. This behavior is equivalent to `--ask-vault-pass` or `--vault-password-file` so is rarely used.

For example, to use a password file `dev-password`:

```
ansible-playbook --vault-id dev-password site.yml
```

To prompt for the password:

```
ansible-playbook --vault-id @prompt site.yml
```

To get the password from an executable script `my-vault-password-client.py`:

```
ansible-playbook --vault-id my-vault-password-client.py
```

1.7.5 Configuring defaults for using encrypted content

Setting a default vault ID

If you use one vault ID more frequently than any other, you can set the config option *DEFAULT_VAULT_IDENTITY_LIST* to specify a default vault ID and password source. Ansible will use the default vault ID and source any time you do not specify `--vault-id`. You can set multiple values for this option. Setting multiple values is equivalent to passing multiple `--vault-id` cli options.

Setting a default password source

If you don't want to provide the password file on the command line or if you use one vault password file more frequently than any other, you can set the *DEFAULT_VAULT_PASSWORD_FILE* config option or the `ANSIBLE_VAULT_PASSWORD_FILE` environment variable to specify a default file to use. For example, if you set `ANSIBLE_VAULT_PASSWORD_FILE=~/.vault_pass.txt`, Ansible will automatically search for the password in that file. This is useful if, for example, you use Ansible from a continuous integration system such as Jenkins.

The file that you reference can be either a file containing the password (in plain text), or it can be a script (with executable permissions set) that returns the password.

1.7.6 When are encrypted files made visible?

In general, content you encrypt with Ansible Vault remains encrypted after execution. However, there is one exception. If you pass an encrypted file as the `src` argument to the copy, template, unarchive, script or assemble module, the file will not be encrypted on the target host (assuming you supply the correct vault password when you run the play). This behavior is intended and useful. You can encrypt a configuration file or template to avoid sharing the details of your configuration, but when you copy that configuration to servers in your environment, you want it to be decrypted so local users and processes can access it.

1.7.7 Format of files encrypted with Ansible Vault

Ansible Vault creates UTF-8 encoded txt files. The file format includes a newline terminated header. For example:

```
$ANSIBLE_VAULT;1.1;AES256
```

or

```
$ANSIBLE_VAULT;1.2;AES256;vault-id-label
```

The header contains up to four elements, separated by semi-colons (`;`).

1. The format ID (`$ANSIBLE_VAULT`). Currently `$ANSIBLE_VAULT` is the only valid format ID. The format ID identifies content that is encrypted with Ansible Vault (via vault.is_encrypted_file()).
2. The vault format version (`1.X`). All supported versions of Ansible will currently default to '1.1' or '1.2' if a labeled vault ID is supplied. The '1.0' format is supported for reading only (and will be converted automatically to the '1.1' format on write). The format version is currently used as an exact string compare only (version numbers are not currently 'compared').
3. The cipher algorithm used to encrypt the data (`AES256`). Currently `AES256` is the only supported cipher algorithm. Vault format 1.0 used 'AES', but current code always uses 'AES256'.
4. The vault ID label used to encrypt the data (optional, `vault-id-label`) For example, if you encrypt a file with `--vault-id dev@prompt`, the vault-id-label is `dev`.

Note: In the future, the header could change. Fields after the format ID and format version depend on the format version, and future vault format versions may add more cipher algorithm options and/or additional fields.

The rest of the content of the file is the 'vaulttext'. The vaulttext is a text armored version of the encrypted ciphertext. Each line is 80 characters wide, except for the last line which may be shorter.

Ansible Vault payload format 1.1 - 1.2

The vaulttext is a concatenation of the ciphertext and a SHA256 digest with the result 'hexlifyied'.

'hexlify' refers to the `hexlify()` method of the Python Standard Library's binascii module.

hexlify()'ed result of:

- hexlify()'ed string of the salt, followed by a newline (`0x0a`)
- hexlify()'ed string of the crypted HMAC, followed by a newline. The HMAC is:
  - a RFC2104 style HMAC
    - inputs are:
      - The AES256 encrypted ciphertext
      - A PBKDF2 key. This key, the cipher key, and the cipher IV are generated from:
      - the salt, in bytes
      - 10000 iterations
      - SHA256() algorithm
      - the first 32 bytes are the cipher key
      - the second 32 bytes are the HMAC key
      - remaining 16 bytes are the cipher IV
- hexlify()'ed string of the ciphertext. The ciphertext is:
- AES256 encrypted data. The data is encrypted using:
  - AES-CTR stream cipher
  - cipher key
  - IV
  - a 128 bit counter block seeded from an integer IV
  - the plaintext
    - the original plaintext
    - padding up to the AES256 blocksize. (The data used for padding is based on RFC5652)

1.8 Using Ansible modules and plugins

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible guide for working with modules, plugins, and collections.

Ansible modules are units of code that can control system resources or execute system commands. Ansible provides a module library that you can execute directly on remote hosts or through playbooks. You can also write custom modules.

Similar to modules are plugins, which are pieces of code that extend core Ansible functionality. Ansible uses a plugin architecture to enable a rich, flexible, and expandable feature set. Ansible ships with several plugins and lets you easily use your own plugins.

1.8.1 Introduction to modules

Modules (also referred to as "task plugins" or "library plugins") are discrete units of code that can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote managed node, and collects return values. In Ansible 2.10 and later, most modules are hosted in collections.

You can execute modules from the command line.

```
ansible webservers -m service -a "name=httpd state=started"
ansible webservers -m ping
ansible webservers -m command -a "/sbin/reboot -t now"
```

Each module supports taking arguments. Nearly all modules take `key=value` arguments, space delimited. Some modules take no arguments, and the command/shell modules simply take the string of the command you want to run.

From playbooks, Ansible modules are executed in a very similar way.

```
- name: reboot the servers
  command: /sbin/reboot -t now
```

Another way to pass arguments to a module is using YAML syntax, also called 'complex args'.

```
- name: restart webserver
  service:
    name: httpd
    state: restarted
```

All modules return JSON format data. This means modules can be written in any programming language. Modules should be idempotent, and should avoid making any changes if they detect that the current state matches the desired final state. When used in an Ansible playbook, modules can trigger 'change events' in the form of notifying *handlers* to run additional tasks.

You can access the documentation for each module from the command line with the ansible-doc tool.

```
ansible-doc yum
```

For a list of all available modules, see the Collection docs, or run the following at a command prompt.

```
ansible-doc -l
```

See also:

Introduction to ad hoc commands
Examples of using modules in /usr/bin/ansible

Working with playbooks
Examples of using modules with /usr/bin/ansible-playbook

Should you develop a module?
How to write your own modules

Python API
Examples of using modules with the Python API

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.8.2 Module maintenance and support

If you are using a module and you discover a bug, you may want to know where to report that bug, who is responsible for fixing it, and how you can track changes to the module. If you are a Red Hat subscriber, you may want to know whether you can get support for the issue you are facing.

Starting in Ansible 2.10, most modules live in collections. The distribution method for each collection reflects the maintenance and support for the modules in that collection.

Maintenance

Collection	Code location	Maintained by
ansible.builtin	ansible/ansible repo on GitHub	core team
distributed on Galaxy	various; follow `repo` link	community or partners
distributed on Automation Hub	various; follow `repo` link	content team or partners

Issue Reporting

If you find a bug that affects a plugin in the main Ansible repo, also known as `ansible-core`:

1. Confirm that you are running the latest stable version of Ansible or the devel branch.
2. Look at the issue tracker in the Ansible repo to see if an issue has already been filed.
3. Create an issue if one does not already exist. Include as much detail as you can about the behavior you discovered.

If you find a bug that affects a plugin in a Galaxy collection:

1. Find the collection on Galaxy.
2. Find the issue tracker for the collection.
3. Look there to see if an issue has already been filed.
4. Create an issue if one does not already exist. Include as much detail as you can about the behavior you discovered.

Some partner collections may be hosted in private repositories.

If you are not sure whether the behavior you see is a bug, if you have questions, if you want to discuss development-oriented topics, or if you just want to get in touch, use one of our Google mailing lists or chat channels (using Matrix at ansible.im or using IRC at irc.libera.chat) to *communicate with Ansiblers*.

If you find a bug that affects a module in an Automation Hub collection:

1. If the collection offers an Issue Tracker link on Automation Hub, click there and open an issue on the collection repository. If it does not, follow the standard process for reporting issues on the Red Hat Customer Portal. You must have a subscription to the Red Hat Ansible Automation Platform to create an issue on the portal.

Support

All plugins that remain in `ansible-core` and all collections hosted in Automation Hub are supported by Red Hat. No other plugins or collections are supported by Red Hat. If you have a subscription to the Red Hat Ansible Automation Platform, you can find more information and resources on the Red Hat Customer Portal.

See also:

Introduction to ad hoc commands
Examples of using modules in /usr/bin/ansible

Working with playbooks
Examples of using modules with /usr/bin/ansible-playbook

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.8.3 Rejecting modules

If you want to avoid using certain modules, you can add them to a reject list to prevent Ansible from loading them. To reject plugins, create a yaml configuration file. The default location for this file is `/etc/ansible/plugin_filters.yml`. You can select a different path for the reject list using the *PLUGIN_FILTERS_CFG* setting in the `defaults` section of your ansible.cfg. Here is an example reject list:

```
---
filter_version: '1.0'
module_rejectlist:
  # Deprecated
  - docker
  # We only allow pip, not easy_install
  - easy_install
```

The file contains two fields:

- A file version so that you can update the format while keeping backwards compatibility in the future. The present version should be the string, `"1.0"`
- A list of modules to reject. Ansible will not load any module in this list when it searches for a module to invoke for a task.

Note: The `stat` module is required for Ansible to run. Do not add this module to your reject list.

1.8.4 Working with plugins

Plugins are pieces of code that augment Ansible's core functionality. Ansible uses a plugin architecture to enable a rich, flexible and expandable feature set.

Ansible ships with a number of handy plugins, and you can easily write your own.

This section covers the various types of plugins that are included with Ansible:

Action plugins

Action plugins act in conjunction with modules to execute the actions required by playbook tasks. They usually execute automatically in the background doing prerequisite work before modules execute.

The 'normal' action plugin is used for modules that do not already have an action plugin. If necessary, you can *create custom action plugins*.

Enabling action plugins

You can enable a custom action plugin by either dropping it into the `action_plugins` directory adjacent to your play, inside a role, or by putting it in one of the action plugin directory sources configured in *ansible.cfg*.

Using action plugins

Action plugin are executed by default when an associated module is used; no action is required.

Plugin list

You cannot list action plugins directly, they show up as their counterpart modules:

Use `ansible-doc -l` to see the list of available modules. Use `ansible-doc <name>` to see specific documentation and examples, this should note if the module has a corresponding action plugin.

See also:

Cache plugins
Cache plugins

Callback plugins
Callback plugins

Connection plugins
Connection plugins

Inventory plugins
Inventory plugins

Shell plugins
Shell plugins

Strategy plugins
Strategy plugins

Vars plugins
Vars plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Become plugins

New in version 2.8.

Become plugins work to ensure that Ansible can use certain privilege escalation systems when running the basic commands to work with the target machine as well as the modules required to execute the tasks specified in the play.

These utilities (`sudo`, `su`, `doas`, and so on) generally let you 'become' another user to execute a command with the permissions of that user.

Enabling Become Plugins

The become plugins shipped with Ansible are already enabled. Custom plugins can be added by placing them into a `become_plugins` directory adjacent to your play, inside a role, or by placing them in one of the become plugin directory sources configured in *ansible.cfg*.

Using Become Plugins

In addition to the default configuration settings in *Ansible Configuration Settings* or the `--become-method` command line option, you can use the `become_method` keyword in a play or, if you need to be 'host specific', the connection variable `ansible_become_method` to select the plugin to use.

You can further control the settings for each plugin via other configuration options detailed in the plugin themselves (linked below).

Plugin List

You can use `ansible-doc -t become -l` to see the list of available plugins. Use `ansible-doc -t become <plugin name>` to see specific documentation and examples.

See also:

Ansible playbooks
An introduction to playbooks

Inventory plugins
Inventory plugins

Callback plugins
Callback plugins

Filter plugins
Filter plugins

Test plugins
Test plugins

Lookup plugins
Lookup plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Cache plugins

Cache plugins allow Ansible to store gathered facts or inventory source data without the performance hit of retrieving them from source.

The default cache plugin is the memory plugin, which only caches the data for the current execution of Ansible. Other plugins with persistent storage are available to allow caching the data across runs. Some of these cache plugins write to files, others write to databases.

You can use different cache plugins for inventory and facts. If you enable inventory caching without setting an inventory-specific cache plugin, Ansible uses the fact cache plugin for both facts and inventory. If necessary, you can *create custom cache plugins*.

Enabling fact cache plugins

Fact caching is always enabled. However, only one fact cache plugin can be active at a time. You can select the cache plugin to use for fact caching in the Ansible configuration, either with an environment variable:

```
export ANSIBLE_CACHE_PLUGIN=jsonfile
```

or in the `ansible.cfg` file:

```
[defaults]
fact_caching=redis
```

If the cache plugin is in a collection use the fully qualified name:

```
[defaults]
fact_caching = namespace.collection_name.cache_plugin_name
```

To enable a custom cache plugin, save it in a `cache_plugins` directory adjacent to your play, inside a role, or in one of the directory sources configured in *ansible.cfg*.

You also need to configure other settings specific to each plugin. Consult the individual plugin documentation or the Ansible *configuration* for more details.

Enabling inventory cache plugins

Inventory caching is disabled by default. To cache inventory data, you must enable inventory caching and then select the specific cache plugin you want to use. Not all inventory plugins support caching, so check the documentation for the inventory plugin(s) you want to use. You can enable inventory caching with an environment variable:

```
export ANSIBLE_INVENTORY_CACHE=True
```

or in the `ansible.cfg` file:

```
[inventory]
cache=True
```

or if the inventory plugin accepts a YAML configuration source, in the configuration file:

```
# dev.aws_ec2.yaml
plugin: aws_ec2
cache: True
```

Only one inventory cache plugin can be active at a time. You can set it with an environment variable:

```
export ANSIBLE_INVENTORY_CACHE_PLUGIN=jsonfile
```

or in the ansible.cfg file:

```
[inventory]
cache_plugin=jsonfile
```

or if the inventory plugin accepts a YAML configuration source, in the configuration file:

```
# dev.aws_ec2.yaml
plugin: aws_ec2
cache_plugin: jsonfile
```

To cache inventory with a custom plugin in your plugin path, follow the *developer guide on cache plugins*.

To cache inventory with a cache plugin in a collection, use the FQCN:

```
[inventory]
cache_plugin=collection_namespace.collection_name.cache_plugin
```

If you enable caching for inventory plugins without selecting an inventory-specific cache plugin, Ansible falls back to caching inventory with the fact cache plugin you configured. Consult the individual inventory plugin documentation or the Ansible *configuration* for more details.

Using cache plugins

Cache plugins are used automatically once they are enabled.

Plugin list

You can use `ansible-doc -t cache -l` to see the list of available plugins. Use `ansible-doc -t cache <plugin name>` to see specific documentation and examples.

See also:

Action plugins
    Action plugins

Callback plugins
    Callback plugins

Connection plugins
    Connection plugins

Inventory plugins
Inventory plugins

Shell plugins
Shell plugins

Strategy plugins
Strategy plugins

Vars plugins
Vars plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Callback plugins

Callback plugins enable adding new behaviors to Ansible when responding to events. By default, callback plugins control most of the output you see when running the command line programs, but can also be used to add additional output, integrate with other tools and marshal the events to a storage backend. If necessary, you can *create custom callback plugins*.

Example callback plugins

The log_plays callback is an example of how to record playbook events to a log file, and the mail callback sends email on playbook failures.

The say callback responds with computer synthesized speech in relation to playbook events.

Enabling callback plugins

You can activate a custom callback by either dropping it into a `callback_plugins` directory adjacent to your play, inside a role, or by putting it in one of the callback directory sources configured in *ansible.cfg*.

Plugins are loaded in alphanumeric order. For example, a plugin implemented in a file named *1_first.py* would run before a plugin file named *2_second.py*.

Most callbacks shipped with Ansible are disabled by default and need to be enabled in your *ansible.cfg* file in order to function. For example:

```
#callbacks_enabled = timer, mail, profile_roles, collection_namespace.collection_name.
↪custom_callback
```

Setting a callback plugin for `ansible-playbook`

You can only have one plugin be the main manager of your console output. If you want to replace the default, you should define `CALLBACK_TYPE = stdout` in the subclass and then configure the stdout plugin in *ansible.cfg*. For example:

```
stdout_callback = dense
```

or for my custom callback:

```
stdout_callback = mycallback
```

This only affects *ansible-playbook* by default.

Setting a callback plugin for ad hoc commands

The *ansible* ad hoc command specifically uses a different callback plugin for stdout, so there is an extra setting in *Ansible Configuration Settings* you need to add to use the stdout callback defined above:

```
[defaults]
bin_ansible_callbacks=True
```

You can also set this as an environment variable:

```
export ANSIBLE_LOAD_CALLBACK_PLUGINS=1
```

Types of callback plugins

There are three types of callback plugins:

stdout callback plugins
These plugins handle the main console output. Only one of these can be active.

aggregate callback plugins
Aggregate callbacks can add additional console output next to a stdout callback. This can be aggregate information at the end of a playbook run, additional per-task output, or anything else.

notification callback plugins
Notification callbacks inform other applications, services, or systems. This can be anything from logging to databases, informing on errors in Instant Messaging applications, or sending emails when a server is unreachable.

Plugin list

You can use `ansible-doc -t callback -l` to see the list of available plugins. Use `ansible-doc -t callback <plugin name>` to see specific documents and examples.

See also:

Action plugins
Action plugins

Cache plugins
Cache plugins

Connection plugins
Connection plugins

Inventory plugins
Inventory plugins

Shell plugins
Shell plugins

Strategy plugins
Strategy plugins

Vars plugins
Vars plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Cliconf plugins

Cliconf plugins are abstractions over the CLI interface to network devices. They provide a standard interface for Ansible to execute tasks on those network devices.

These plugins generally correspond one-to-one to network device platforms. Ansible loads the appropriate cliconf plugin automatically based on the `ansible_network_os` variable.

Adding cliconf plugins

You can extend Ansible to support other network devices by dropping a custom plugin into the `cliconf_plugins` directory.

Using cliconf plugins

The cliconf plugin to use is determined automatically from the `ansible_network_os` variable. There should be no reason to override this functionality.

Most cliconf plugins can operate without configuration. A few have additional options that can be set to affect how tasks are translated into CLI commands.

Plugins are self-documenting. Each plugin should document its configuration options.

Viewing cliconf plugins

These plugins have migrated to collections on Ansible Galaxy. If you installed Ansible version 2.10 or later using `pip`, you have access to several cliconf plugins. To list all available cliconf plugins on your control node, type `ansible-doc -t cliconf -l`. To view plugin-specific documentation and examples, use `ansible-doc -t cliconf`.

See also:

Ansible for Network Automation
An overview of using Ansible to automate networking devices.

User Mailing List
Have a question? Stop by the google group!

irc.libera.chat
#ansible-network IRC chat channel

Connection plugins

Connection plugins allow Ansible to connect to the target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time.

By default, Ansible ships with several connection plugins. The most commonly used are the paramiko SSH, native ssh (just called ssh), and local connection types. All of these can be used in playbooks and with **`/usr/bin/ansible`** to decide how you want to talk to remote machines. If necessary, you can *create custom connection plugins*.

The basics of these connection types are covered in the getting started section.

`ssh` plugins

Because ssh is the default protocol used in system administration and the protocol most used in Ansible, ssh options are included in the command line tools. See *ansible-playbook* for more details.

Adding connection plugins

You can extend Ansible to support other transports (such as SNMP or message bus) by dropping a custom plugin into the `connection_plugins` directory.

Using connection plugins

You can set the connection plugin globally via *configuration*, at the command line (`-c`, `--connection`), as a *keyword* in your play, or by setting a *variable*, most often in your inventory. For example, for Windows machines you might want to set the winrm plugin as an inventory variable.

Most connection plugins can operate with minimal configuration. By default they use the inventory hostname and defaults to find the target host.

Plugins are self-documenting. Each plugin should document its configuration options. The following are connection variables common to most connection plugins:

ansible_host
The name of the host to connect to, if different from the *inventory* hostname.

ansible_port
The ssh port number, for ssh and paramiko_ssh it defaults to 22.

ansible_user
The default user name to use for log in. Most plugins default to the 'current user running Ansible'.

Each plugin might also have a specific version of a variable that overrides the general version. For example, `ansible_ssh_host` for the ssh plugin.

Plugin list

You can use `ansible-doc -t connection -l` to see the list of available plugins. Use `ansible-doc -t connection <plugin name>` to see detailed documentation and examples.

See also:

Working with Playbooks
An introduction to playbooks

Callback plugins
Callback plugins

Filter plugins
Filter plugins

Test plugins
Test plugins

Lookup plugins
Lookup plugins

Vars plugins
    Vars plugins

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Docs fragments

Docs fragments allow you to document common parameters of multiple plugins or modules in a single place.

Enabling docs fragments

You can add a custom docs fragment by dropping it into a `doc_fragments` directory adjacent to your collection or role, just like any other plugin.

Using docs fragments

Only collection developers and maintainers use docs fragments. For more information on using docs fragments, see *Documentation fragments* or *Using documentation fragments in collections*.

See also:

Developing modules
    An introduction to creating Ansible modules

Developing collections
    An guide to creating Ansible collections

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Filter plugins

Filter plugins manipulate data. With the right filter you can extract a particular value, transform data types and formats, perform mathematical calculations, split and concatenate strings, insert dates and times, and do much more. Ansible uses the standard filters shipped with Jinja2 and adds some specialized filter plugins. You can *create custom Ansible filters as plugins*.

Enabling filter plugins

You can add a custom filter plugin by dropping it into a `filter_plugins` directory adjacent to your play, inside a role, or by putting it in one of the filter plugin directory sources configured in *ansible.cfg*.

Using filter plugins

You can use filters anywhere you can use templating in Ansible: in a play, in variables file, or in a Jinja2 template for the template module. For more information on using filter plugins, see *Using filters to manipulate data*. Filters can return any type of data, but if you want to always return a boolean (`True` or `False`) you should be looking at a test instead.

```
vars:
  yaml_string: "{{ some_variable|to_yaml }}"
```

Filters are the preferred way to manipulate data in Ansible, you can identify a filter because it is normally preceded by a `|`, with the expression on the left of it being the first input of the filter. Additional parameters may be passed into the filter itself as you would to most programming functions. These parameters can be either `positional` (passed in order) or `named` (passed as key=value pairs). When passing both types, positional arguments should go first.

```
passing_positional: {{ (x == 32) | ternary('x is 32', 'x is not 32') }}
passing_extra_named_parameters: {{ some_variable | to_yaml(indent=8, width=1337) }}
passing_both: {{ some_variable| ternary('true value', 'false value', none_val='NULL') }}
```

In the documentation, filters will always have a C(_input) option that corresponds to the expression to the left of c(|). A C(positional:) field in the documentation will show which options are positional and in which order they are required.

Plugin list

You can use `ansible-doc -t filter -l` to see the list of available plugins. Use `ansible-doc -t filter <plugin name>` to see specific documents and examples.

See also:

Ansible playbooks
An introduction to playbooks

Inventory plugins
Inventory plugins

Callback plugins
Callback plugins

Test plugins
Test plugins

Lookup plugins
Lookup plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Httpapi plugins

Httpapi plugins tell Ansible how to interact with a remote device's HTTP-based API and execute tasks on the device.

Each plugin represents a particular dialect of API. Some are platform-specific (Arista eAPI, Cisco NXAPI), while others might be usable on a variety of platforms (RESTCONF). Ansible loads the appropriate httpapi plugin automatically based on the `ansible_network_os` variable.

Adding httpapi plugins

You can extend Ansible to support other APIs by dropping a custom plugin into the `httpapi_plugins` directory. See *Developing httpapi plugins* for details.

Using httpapi plugins

The httpapi plugin to use is determined automatically from the `ansible_network_os` variable.

Most httpapi plugins can operate without configuration. Additional options may be defined by each plugin.

Plugins are self-documenting. Each plugin should document its configuration options.

The following sample playbook shows the httpapi plugin for an Arista network device, assuming an inventory variable set as `ansible_network_os=eos` for the httpapi plugin to trigger off:

```
- hosts: leaf01
  connection: httpapi
  gather_facts: false
  tasks:

    - name: type a simple arista command
      eos_command:
        commands:
          - show version | json
      register: command_output

    - name: print command output to terminal window
      debug:
        var: command_output.stdout[0]["version"]
```

See the full working example on GitHub.

Viewing httpapi plugins

These plugins have migrated to collections on Ansible Galaxy. If you installed Ansible version 2.10 or later using `pip`, you have access to several httpapi plugins. To list all available httpapi plugins on your control node, type `ansible-doc -t httpapi -l`. To view plugin-specific documentation and examples, use `ansible-doc -t httpapi`.

See also:

Ansible for Network Automation
An overview of using Ansible to automate networking devices.

Developing network modules
How to develop network modules.

User Mailing List
Have a question? Stop by the google group!

irc.libera.chat
#ansible-network IRC chat channel

Inventory plugins

Inventory plugins allow users to point at data sources to compile the inventory of hosts that Ansible uses to target tasks, either using the `-i /path/to/file` and/or `-i 'host1, host2'` command line parameters or from other configuration sources. If necessary, you can *create custom inventory plugins*.

Enabling inventory plugins

Most inventory plugins shipped with Ansible are enabled by default or can be used by with the `auto` plugin.

In some circumstances, for example, if the inventory plugin does not use a YAML configuration file, you may need to enable the specific plugin. You can do this by setting `enable_plugins` in your *ansible.cfg* file in the `[inventory]` section. Modifying this will override the default list of enabled plugins. Here is the default list of enabled plugins that ships with Ansible:

```
[inventory]
enable_plugins = host_list, script, auto, yaml, ini, toml
```

If the plugin is in a collection and is not being picked up by the *auto* statement, you can append the fully qualified name:

```
[inventory]
enable_plugins = host_list, script, auto, yaml, ini, toml, namespace.collection_name.
↪inventory_plugin_name
```

Or, if it is a local plugin, perhaps stored in the path set by *DEFAULT_INVENTORY_PLUGIN_PATH*, you could reference it as follows:

```
[inventory]
enable_plugins = host_list, script, auto, yaml, ini, toml, my_plugin
```

If you use a plugin that supports a YAML configuration source, make sure that the name matches the name provided in the plugin entry of the inventory source file.

Using inventory plugins

To use an inventory plugin, you must provide an inventory source. Most of the time this is a file containing host information or a YAML configuration file with options for the plugin. You can use the -i flag to provide inventory sources or configure a default inventory path.

```
ansible hostname -i inventory_source -m ansible.builtin.ping
```

To start using an inventory plugin with a YAML configuration source, create a file with the accepted filename schema documented for the plugin in question, then add plugin: plugin_name. Use the fully qualified name if the plugin is in a collection.

```
# demo.aws_ec2.yml
plugin: amazon.aws.aws_ec2
```

Each plugin should document any naming restrictions. In addition, the YAML config file must end with the extension yml or yaml to be enabled by default with the auto plugin (otherwise, see the section above on enabling plugins).

After providing any required options, you can view the populated inventory with ansible-inventory -i demo.aws_ec2.yml --graph:

```
@all:
  |--@aws_ec2:
  |  |--ec2-12-345-678-901.compute-1.amazonaws.com
  |  |--ec2-98-765-432-10.compute-1.amazonaws.com
  |--@ungrouped:
```

If you are using an inventory plugin in a playbook-adjacent collection and want to test your setup with ansible-inventory, use the --playbook-dir flag.

Your inventory source might be a directory of inventory configuration files. The constructed inventory plugin only operates on those hosts already in inventory, so you may want the constructed inventory configuration parsed at a particular point (such as last). Ansible parses the directory recursively, alphabetically. You cannot configure the parsing approach, so name your files to make it work predictably. Inventory plugins that extend constructed features directly can work around that restriction by adding constructed options in addition to the inventory plugin options. Otherwise, you can use -i with multiple sources to impose a specific order, for example -i demo.aws_ec2.yml -i clouds.yml -i constructed.yml.

You can create dynamic groups using host variables with the constructed keyed_groups option. The option groups can also be used to create groups and compose creates and modifies host variables. Here is an aws_ec2 example utilizing constructed features:

```
# demo.aws_ec2.yml
plugin: amazon.aws.aws_ec2
regions:
  - us-east-1
  - us-east-2
keyed_groups:
  # add hosts to tag_Name_value groups for each aws_ec2 host's tags.Name variable
  - key: tags.Name
    prefix: tag_Name_
```

(continues on next page)

(continued from previous page)

```
    separator: ""
  # If you have a tag called "Role" which has the value "Webserver", this will add the␣
↪group
  # role_Webserver and add any hosts that have that tag assigned to it.
  - key: tags.Role
    prefix: role
groups:
  # add hosts to the group development if any of the dictionary's keys or values is the␣
↪word 'devel'
  development: "'devel' in (tags|list)"
  # add hosts to the "private_only" group if the host doesn't have a public IP associated␣
↪to it
  private_only: "public_ip_address is not defined"
compose:
  # use a private address where a public one isn't assigned
  ansible_host: public_ip_address|default(private_ip_address)
  # alternatively, set the ansible_host variable to connect with the private IP address␣
↪without changing the hostname
  # ansible_host: private_ip_address
  # if you *must* set a string here (perhaps to identify the inventory source if you␣
↪have multiple
  # accounts you want to use as sources), you need to wrap this in two sets of quotes,␣
↪either ' then "
  # or " then '
  some_inventory_wide_string: '"Yes, you need both types of quotes here"'
```

Now the output of `ansible-inventory -i demo.aws_ec2.yml --graph`:

```
@all:
  |--@aws_ec2:
  |  |--ec2-12-345-678-901.compute-1.amazonaws.com
  |  |--ec2-98-765-432-10.compute-1.amazonaws.com
  |  |--...
  |--@development:
  |  |--ec2-12-345-678-901.compute-1.amazonaws.com
  |  |--ec2-98-765-432-10.compute-1.amazonaws.com
  |--@role_Webserver
  |  |--ec2-12-345-678-901.compute-1.amazonaws.com
  |--@tag_Name_ECS_Instance:
  |  |--ec2-98-765-432-10.compute-1.amazonaws.com
  |--@tag_Name_Test_Server:
  |  |--ec2-12-345-678-901.compute-1.amazonaws.com
  |--@ungrouped
```

If a host does not have the variables in the configuration above (in other words, `tags.Name`, `tags`, `private_ip_address`), the host will not be added to groups other than those that the inventory plugin creates and the `ansible_host` host variable will not be modified.

Inventory plugins that support caching can use the general settings for the fact cache defined in the `ansible.cfg` file's `[defaults]` section or define inventory-specific settings in the `[inventory]` section. Individual plugins can define plugin-specific cache settings in their config file:

```
# demo.aws_ec2.yml
```

(continues on next page)

(continued from previous page)

```yaml
plugin: amazon.aws.aws_ec2
cache: true
cache_plugin: ansible.builtin.jsonfile
cache_timeout: 7200
cache_connection: /tmp/aws_inventory
cache_prefix: aws_ec2
```

Here is an example of setting inventory caching with some fact caching defaults for the cache plugin used and the timeout in an `ansible.cfg` file:

```ini
[defaults]
fact_caching = ansible.builtin.jsonfile
fact_caching_connection = /tmp/ansible_facts
cache_timeout = 3600

[inventory]
cache = yes
cache_connection = /tmp/ansible_inventory
```

Plugin list

You can use `ansible-doc -t inventory -l` to see the list of available plugins. Use `ansible-doc -t inventory <plugin name>` to see plugin-specific documentation and examples.

See also:

Ansible playbooks
An introduction to playbooks

Callback plugins
Callback plugins

Connection plugins
Connection plugins

Filter plugins
Filter plugins

Test plugins
Test plugins

Lookup plugins
Lookup plugins

Vars plugins
Vars plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Lookup plugins

Lookup plugins are an Ansible-specific extension to the Jinja2 templating language. You can use lookup plugins to access data from outside sources (files, databases, key/value stores, APIs, and other services) within your playbooks. Like all *templating*, lookups execute and are evaluated on the Ansible control machine. Ansible makes the data returned by a lookup plugin available using the standard templating system. You can use lookup plugins to load variables or templates with information from external sources. You can *create custom lookup plugins*.

Note:

- Lookups are executed with a working directory relative to the role or play, as opposed to local tasks, which are executed relative the executed script.
- Pass `wantlist=True` to lookups to use in Jinja2 template "for" loops.
- By default, lookup return values are marked as unsafe for security reasons. If you trust the outside source your lookup accesses, pass `allow_unsafe=True` to allow Jinja2 templates to evaluate lookup values.

Warning:

- Some lookups pass arguments to a shell. When using variables from a remote/untrusted source, the *|quote* filter to ensure safe usage.

Enabling lookup plugins

Ansible enables all lookup plugins it can find. You can activate a custom lookup by either dropping it into a `lookup_plugins` directory adjacent to your play, inside the `plugins/lookup/` directory of a collection you have installed, inside a standalone role, or in one of the lookup directory sources configured in *ansible.cfg*.

Using lookup plugins

You can use lookup plugins anywhere you can use templating in Ansible: in a play, in variables file, or in a Jinja2 template for the template module. For more information on using lookup plugins, see *Lookups*.

```
vars:
  file_contents: "{{ lookup('file', 'path/to/file.txt') }}"
```

Lookups are an integral part of loops. Wherever you see `with_`, the part after the underscore is the name of a lookup. For this reason, lookups are expected to output lists; for example, `with_items` uses the items lookup:

```
tasks:
  - name: count to 3
```

(continues on next page)

(continued from previous page)

```
    debug: msg={{ item }}
    with_items: [1, 2, 3]
```

You can combine lookups with *filters*, *tests* and even each other to do some complex data generation and manipulation. For example:

```
tasks:
  - name: valid but useless and over complicated chained lookups and filters
    debug: msg="find the answer here:\n{{ lookup('url', 'https://google.com/search/?q='␣
↪+ item|urlencode)|join(' ') }}"
    with_nested:
      - "{{ lookup('consul_kv', 'bcs/' + lookup('file', '/the/question') + ',␣
↪host=localhost, port=2000')|shuffle }}"
      - "{{ lookup('sequence', 'end=42 start=2 step=2')|map('log', 4)|list) }}"
      - ['a', 'c', 'd', 'c']
```

New in version 2.6.

You can control how errors behave in all lookup plugins by setting `errors` to `ignore`, `warn`, or `strict`. The default setting is `strict`, which causes the task to fail if the lookup returns an error. For example:

To ignore lookup errors:

```
- name: if this file does not exist, I do not care .. file plugin itself warns anyway ...
  debug: msg="{{ lookup('file', '/nosuchfile', errors='ignore') }}"
```

```
[WARNING]: Unable to find '/nosuchfile' in expected paths (use -vvvvv to see paths)

ok: [localhost] => {
    "msg": ""
}
```

To get a warning instead of a failure:

```
- name: if this file does not exist, let me know, but continue
  debug: msg="{{ lookup('file', '/nosuchfile', errors='warn') }}"
```

```
[WARNING]: Unable to find '/nosuchfile' in expected paths (use -vvvvv to see paths)

[WARNING]: An unhandled exception occurred while running the lookup plugin 'file'. Error␣
↪was a <class 'ansible.errors.AnsibleError'>, original message: could not locate file␣
↪in lookup: /nosuchfile

ok: [localhost] => {
    "msg": ""
}
```

To get a fatal error (the default):

```
- name: if this file does not exist, FAIL (this is the default)
  debug: msg="{{ lookup('file', '/nosuchfile', errors='strict') }}"
```

```
[WARNING]: Unable to find '/nosuchfile' in expected paths (use -vvvvv to see paths)

fatal: [localhost]: FAILED! => {"msg": "An unhandled exception occurred while running␣
↪the lookup plugin 'file'. Error was a <class 'ansible.errors.AnsibleError'>, original␣
↪message: could not locate file in lookup: /nosuchfile"}
```

Forcing lookups to return lists: `query` and `wantlist=True`

New in version 2.5.

In Ansible 2.5, a new Jinja2 function called `query` was added for invoking lookup plugins. The difference between `lookup` and `query` is largely that `query` will always return a list. The default behavior of `lookup` is to return a string of comma separated values. `lookup` can be explicitly configured to return a list using `wantlist=True`.

This feature provides an easier and more consistent interface for interacting with the new `loop` keyword, while maintaining backwards compatibility with other uses of `lookup`.

The following examples are equivalent:

```
lookup('dict', dict_variable, wantlist=True)

query('dict', dict_variable)
```

As demonstrated above, the behavior of `wantlist=True` is implicit when using `query`.

Additionally, `q` was introduced as a shortform of `query`:

```
q('dict', dict_variable)
```

Plugin list

You can use `ansible-doc -t lookup -l` to see the list of available plugins. Use `ansible-doc -t lookup <plugin name>` to see specific documents and examples.

See also:

Ansible playbooks
    An introduction to playbooks

Inventory plugins
    Ansible inventory plugins

Callback plugins
    Ansible callback plugins

Filter plugins
    Jinja2 filter plugins

Test plugins
    Jinja2 test plugins

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

Modules

Modules are the main building blocks of Ansible playbooks. Although we do not generally speak of "module plugins", a module is a type of plugin. For a developer-focused description of the differences between modules and other plugins, see *Modules and plugins: what is the difference?*.

Enabling modules

You can enable a custom module by dropping it into one of these locations:

- any directory added to the `ANSIBLE_LIBRARY` environment variable (`$ANSIBLE_LIBRARY` takes a colon-separated list like `$PATH`)
- `~/.ansible/plugins/modules/`
- `/usr/share/ansible/plugins/modules/`

For more information on using local custom modules, see *Adding a module or plugin outside of a collection*.

Using modules

For information on using modules in ad hoc tasks, see *Introduction to ad hoc commands*. For information on using modules in playbooks, see *Ansible playbooks*.

See also:

Ansible playbooks
An introduction to playbooks

Developing modules
An introduction to creating Ansible modules

Developing collections
An guide to creating Ansible collections

User Mailing List
Have a question? Stop by the google group!

irc.libera.chat
#ansible-devel IRC chat channel

Module utilities

Module utilities contain shared code used by multiple plugins. You can write *custom module utilities*.

Enabling module utilities

You can add a custom module utility by dropping it into a `module_utils` directory adjacent to your collection or role, just like any other plugin.

Using module utilities

For information on using module utilities, see *Using and developing module utilities*.

See also:

Developing modules
An introduction to creating Ansible modules

Developing collections
An guide to creating Ansible collections

User Mailing List
Have a question? Stop by the google group!

irc.libera.chat
#ansible-devel IRC chat channel

Netconf plugins

Netconf plugins are abstractions over the Netconf interface to network devices. They provide a standard interface for Ansible to execute tasks on those network devices.

These plugins generally correspond one-to-one to network device platforms. Ansible loads the appropriate netconf plugin automatically based on the `ansible_network_os` variable. If the platform supports standard Netconf implementation as defined in the Netconf RFC specification, Ansible loads the `default` netconf plugin. If the platform supports propriety Netconf RPCs, Ansible loads the platform-specific netconf plugin.

Adding netconf plugins

You can extend Ansible to support other network devices by dropping a custom plugin into the `netconf_plugins` directory.

Using netconf plugins

The netconf plugin to use is determined automatically from the `ansible_network_os` variable. There should be no reason to override this functionality.

Most netconf plugins can operate without configuration. A few have additional options that can be set to affect how tasks are translated into netconf commands. A ncclient device specific handler name can be set in the netconf plugin or else the value of `default` is used as per ncclient device handler.

Plugins are self-documenting. Each plugin should document its configuration options.

Listing netconf plugins

These plugins have migrated to collections on Ansible Galaxy. If you installed Ansible version 2.10 or later using `pip`, you have access to several netconf plugins. To list all available netconf plugins on your control node, type `ansible-doc -t netconf -l`. To view plugin-specific documentation and examples, use `ansible-doc -t netconf`.

See also:

Ansible for Network Automation
An overview of using Ansible to automate networking devices.

User Mailing List
Have a question? Stop by the google group!

irc.libera.chat
#ansible-network IRC chat channel

Shell plugins

Shell plugins work to ensure that the basic commands Ansible runs are properly formatted to work with the target machine and allow the user to configure certain behaviors related to how Ansible executes tasks.

Enabling shell plugins

You can add a custom shell plugin by dropping it into a `shell_plugins` directory adjacent to your play, inside a role, or by putting it in one of the shell plugin directory sources configured in *ansible.cfg*.

Warning: You should not alter which plugin is used unless you have a setup in which the default `/bin/sh` is not a POSIX compatible shell or is not available for execution.

Using shell plugins

In addition to the default configuration settings in *Ansible Configuration Settings*, you can use the connection variable *ansible_shell_type* to select the plugin to use. In this case, you will also want to update the *ansible_shell_executable* to match.

You can further control the settings for each plugin via other configuration options detailed in the plugin themselves (linked below).

See also:

Ansible playbooks
An introduction to playbooks

Inventory plugins
Inventory plugins

Callback plugins
Callback plugins

Filter plugins
Filter plugins

Test plugins
Test plugins

Lookup plugins
Lookup plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Strategy plugins

Strategy plugins control the flow of play execution by handling task and host scheduling. For more information on using strategy plugins and other ways to control execution order, see *Controlling playbook execution: strategies and more*.

Enabling strategy plugins

All strategy plugins shipped with Ansible are enabled by default. You can enable a custom strategy plugin by putting it in one of the lookup directory sources configured in *ansible.cfg*.

Using strategy plugins

Only one strategy plugin can be used in a play, but you can use different ones for each play in a playbook or ansible run. By default Ansible uses the linear plugin. You can change this default in Ansible *configuration* using an environment variable:

```
export ANSIBLE_STRATEGY=free
```

or in the *ansible.cfg* file:

```
[defaults]
strategy=linear
```

You can also specify the strategy plugin in the play via the *strategy keyword* in a play:

```
- hosts: all
  strategy: debug
  tasks:
    - copy: src=myhosts dest=/etc/hosts
      notify: restart_tomcat

    - package: name=tomcat state=present

  handlers:
    - name: restart_tomcat
      service: name=tomcat state=restarted
```

Plugin list

You can use `ansible-doc -t strategy -l` to see the list of available plugins. Use `ansible-doc -t strategy <plugin name>` to see plugin-specific specific documentation and examples.

See also:

Ansible playbooks
An introduction to playbooks

Inventory plugins
Inventory plugins

Callback plugins
Callback plugins

Filter plugins
Filter plugins

Test plugins
Test plugins

Lookup plugins
Lookup plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Terminal plugins

Terminal plugins contain information on how to prepare a particular network device's SSH shell is properly initialized to be used with Ansible. This typically includes disabling automatic paging, detecting errors in output, and enabling privileged mode if supported and required on the device.

These plugins correspond one-to-one to network device platforms. Ansible loads the appropriate terminal plugin automatically based on the `ansible_network_os` variable.

Adding terminal plugins

You can extend Ansible to support other network devices by dropping a custom plugin into the `terminal_plugins` directory.

Using terminal plugins

Ansible determines which terminal plugin to use automatically from the `ansible_network_os` variable. There should be no reason to override this functionality.

Terminal plugins operate without configuration. All options to control the terminal are exposed in the `network_cli` connection plugin.

Plugins are self-documenting. Each plugin should document its configuration options.

Viewing terminal plugins

These plugins have migrated to collections on Ansible Galaxy. If you installed Ansible version 2.10 or later using `pip`, you have access to several terminal plugins. To list all available terminal plugins on your control node, type `ansible-doc -t terminal -l`. To view plugin-specific documentation and examples, use `ansible-doc -t terminal`.

See also:

Ansible for Network Automation
An overview of using Ansible to automate networking devices.

Connection plugins
Connection plugins

User Mailing List
Have a question? Stop by the google group!

irc.libera.chat
#ansible-network IRC chat channel

Test plugins

Test plugins evaluate template expressions and return True or False. With test plugins you can create *conditionals* to implement the logic of your tasks, blocks, plays, playbooks, and roles. Ansible uses the *standard tests `_ shipped as part of Jinja, and adds some specialized test plugins. You can :ref:`create custom Ansible test plugins <developing_test_plugins>*.

Enabling test plugins

You can add a custom test plugin by dropping it into a `test_plugins` directory adjacent to your play, inside a role, or by putting it in one of the test plugin directory sources configured in *ansible.cfg*.

Using test plugins

You can use tests anywhere you can use templating in Ansible: in a play, in variables file, or in a Jinja2 template for the template module. For more information on using test plugins, see *Tests*.

Tests always return `True` or `False`, they are always a boolean, if you need a different return type, you should be looking at filters.

You can recognize test plugins by the use of the `is` statement in a template, they can also be used as part of the `select` family of filters.

```
vars:
  is_ready: '{{ task_result is success }}'

tasks:
- name: conditionals are always in 'template' context
  action: dostuff
  when: task_result is failed
```

Tests will always have an `_input` and this is normally what is on the left side of `is`. Tests can also take additional parameters as you would to most programming functions. These parameters can be either `positional` (passed in order) or `named` (passed as key=value pairs). When passing both types, positional arguments should go first.

```
tasks:
- name: pass positional parameter to match test
  action: dostuff
  when: myurl is match("https://example.com/users/.*/resources")

- name: pass named parameter to truthy test
  action: dostuff
  when: myvariable is truthy(convert_bool=True)

- name: pass both types to 'version' test
```

(continues on next page)

(continued from previous page)

```
  action: dostuff
  when: sample_semver_var is version('2.0.0-rc.1+build.123', 'lt', version_type='semver')
```

Using test plugins with lists

As mentioned above, one way to use tests is with the `select` family of filters (`select`, `reject`, `selectattr`, `rejectattr`).

```
# give me only defined variables from a list of variables, using 'defined' test
good_vars: "{{ all_vars|select('defined') }}"

# this uses the 'equalto' test to filter out non 'fixed' type of addresses from a list
only_fixed_addresses:  "{{ all_addresses|selectattr('type', 'equalsto', 'fixed') }}"

# this does the opposite of the previous one
only_fixed_addresses:  "{{ all_addresses|rejectattr('type', 'equalsto', 'fixed') }}"
```

Plugin list

You can use `ansible-doc -t filter -l` to see the list of available plugins. Use `ansible-doc -t filter <plugin name>` to see specific documents and examples.

See also:

Ansible playbooks
An introduction to playbooks

Tests
Using tests

Conditionals
Using conditional statements

Filter plugins
Filter plugins

Tests
Using tests

Lookup plugins
Lookup plugins

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

Vars plugins

Vars plugins inject additional variable data into Ansible runs that did not come from an inventory source, playbook, or command line. Playbook constructs like 'host_vars' and 'group_vars' work using vars plugins. For more details about variables in Ansible, see *Using Variables*.

Vars plugins were partially implemented in Ansible 2.0 and rewritten to be fully implemented starting with Ansible 2.4.

The host_group_vars plugin shipped with Ansible enables reading variables from *Assigning a variable to one machine: host variables* and *Assigning a variable to many machines: group variables*.

Enabling vars plugins

You can activate a custom vars plugin by either dropping it into a `vars_plugins` directory adjacent to your play, inside a role, or by putting it in one of the directory sources configured in *ansible.cfg*.

Most vars plugins are disabled by default. To enable a vars plugin, set `vars_plugins_enabled` in the `defaults` section of *ansible.cfg* or set the `ANSIBLE_VARS_ENABLED` environment variable to the list of vars plugins you want to execute. By default, the host_group_vars plugin shipped with Ansible is enabled.

Starting in Ansible 2.10, you can use vars plugins in collections. All vars plugins in collections must be explicitly enabled and must use the fully qualified collection name in the format `namespace.collection_name.vars_plugin_name`.

```
[defaults]
vars_plugins_enabled = host_group_vars,namespace.collection_name.vars_plugin_name
```

Using vars plugins

By default, vars plugins are used on demand automatically after they are enabled.

Starting in Ansible 2.10, vars plugins can be made to run at specific times. *ansible-inventory* does not use these settings, and always loads vars plugins.

The global setting `RUN_VARS_PLUGINS` can be set in `ansible.cfg` using `run_vars_plugins` in the `defaults` section or by the `ANSIBLE_RUN_VARS_PLUGINS` environment variable. The default option, `demand`, runs any enabled vars plugins relative to inventory sources whenever variables are demanded by tasks. You can use the option `start` to run any enabled vars plugins relative to inventory sources after importing that inventory source instead.

You can also control vars plugin execution on a per-plugin basis for vars plugins that support the `stage` option. To run the host_group_vars plugin after importing inventory you can add the following to *ansible.cfg*:

```
[vars_host_group_vars]
stage = inventory
```

Plugin list

You can use `ansible-doc -t vars -l` to see the list of available vars plugins. Use `ansible-doc -t vars <plugin name>` to see plugin-specific documentation and examples.

See also:

Cache plugins
    Cache plugins

Lookup plugins
    Lookup plugins

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

See also:

Rejecting modules
    Controlling access to modules

Ansible Configuration Settings
    Ansible configuration documentation and settings

Working with command line tools
    Ansible tools, description and options

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

1.8.5 Modules and plugins index

You can find an index of modules and plugins at *Indexes of all modules and plugins*.

1.9 Using Ansible collections

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible guide for working with collections.

Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. You can install and use collections through a distribution server, such as Ansible Galaxy, or a Pulp 3 Galaxy server.

1.9.1 Installing collections

Note: If you install a collection manually as described in this paragraph, the collection will not be upgraded automatically when you upgrade the `ansible` package or `ansible-core`.

Installing collections with `ansible-galaxy`

By default, `ansible-galaxy collection install` uses https://galaxy.ansible.com as the Galaxy server (as listed in the `ansible.cfg` file under *GALAXY_SERVER*). You do not need any further configuration.

See *Configuring the ansible-galaxy client* if you are using any other Galaxy server, such as Red Hat Automation Hub.

To install a collection hosted in Galaxy:

```
ansible-galaxy collection install my_namespace.my_collection
```

To upgrade a collection to the latest available version from the Galaxy server you can use the `--upgrade` option:

```
ansible-galaxy collection install my_namespace.my_collection --upgrade
```

You can also directly use the tarball from your build:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./
↪collections
```

You can build and install a collection from a local source directory. The `ansible-galaxy` utility builds the collection using the `MANIFEST.json` or `galaxy.yml` metadata in the directory.

```
ansible-galaxy collection install /path/to/collection -p ./collections
```

You can also install multiple collections in a namespace directory.

```
ns/
├── collection1/
│   ├── MANIFEST.json
│   └── plugins/
└── collection2/
    ├── galaxy.yml
    └── plugins/
```

```
ansible-galaxy collection install /path/to/ns -p ./collections
```

Note: The install command automatically appends the path `ansible_collections` to the one specified with the `-p` option unless the parent directory is already in a folder called `ansible_collections`.

When using the `-p` option to specify the install path, use one of the values configured in *COLLECTIONS_PATHS*, as this is where Ansible itself will expect to find collections. If you don't specify a path, `ansible-galaxy collection install` installs the collection to the first path defined in *COLLECTIONS_PATHS*, which by default is `~/.ansible/collections`

You can also keep a collection adjacent to the current playbook, under a `collections/ansible_collections/` directory structure.

```
./
├── play.yml
├── collections/
│   └── ansible_collections/
│               └── my_namespace/
│                   └── my_collection/<collection structure lives here>
```

See *Collection structure* for details on the collection directory structure.

Installing collections with signature verification

If a collection has been signed by a *distribution server*, the server will provide ASCII armored, detached signatures to verify the authenticity of the MANIFEST.json before using it to verify the collection's contents. This option is not available on all distribution servers. See *Distributing collections* for a table listing which servers support collection signing.

To use signature verification for signed collections:

1. *Configured a GnuPG keyring* for ansible-galaxy, or provide the path to the keyring with the --keyring option when you install the signed collection.
2. Import the public key from the distribution server into that keyring.

   ```
   gpg --import --no-default-keyring --keyring ~/.ansible/pubring.kbx my-public-key.asc
   ```

3. Verify the signature when you install the collection.

   ```
   ansible-galaxy collection install my_namespace.my_collection --keyring ~/.ansible/
   ↪pubring.kbx
   ```

   The --keyring option is not necessary if you have *configured a GnuPG keyring*.

4. Optionally, verify the signature at any point after installation to prove the collection has not been tampered with. See *Verifying signed collections* for details.

You can also include signatures in addition to those provided by the distribution server. Use the --signature option to verify the collection's MANIFEST.json with these additional signatures. Supplemental signatures should be provided as URIs.

```
ansible-galaxy collection install my_namespace.my_collection --signature https://
↪examplehost.com/detached_signature.asc --keyring ~/.ansible/pubring.kbx
```

GnuPG verification only occurs for collections installed from a distribution server. User-provided signatures are not used to verify collections installed from git repositories, source directories, or URLs/paths to tar.gz files.

You can also include additional signatures in the collection requirements.yml file under the signatures key.

```
# requirements.yml
collections:
  - name: ns.coll
    version: 1.0.0
    signatures:
      - https://examplehost.com/detached_signature.asc
      - file:///path/to/local/detached_signature.asc
```

See *collection requirements file* for details on how to install collections with this file.

By default, verification is considered successful if a minimum of 1 signature successfully verifies the collection. The number of required signatures can be configured with `--required-valid-signature-count` or *GALAXY_REQUIRED_VALID_SIGNATURE_COUNT*. All signatures can be required by setting the option to `all`. To fail signature verification if no valid signatures are found, prepend the value with `+`, such as `+all` or `+1`.

```
export ANSIBLE_GALAXY_GPG_KEYRING=~/.ansible/pubring.kbx
export ANSIBLE_GALAXY_REQUIRED_VALID_SIGNATURE_COUNT=2
ansible-galaxy collection install my_namespace.my_collection --signature https://
↪examplehost.com/detached_signature.asc --signature file:///path/to/local/detached_
↪signature.asc
```

Certain GnuPG errors can be ignored with `--ignore-signature-status-code` or *GALAXY_REQUIRED_VALID_SIGNATURE_COUNT*. *GALAXY_REQUIRED_VALID_SIGNATURE_COUNT* should be a list, and `--ignore-signature-status-code` can be provided multiple times to ignore multiple additional error status codes.

This example requires any signatures provided by the distribution server to verify the collection except if they fail due to NO_PUBKEY:

```
export ANSIBLE_GALAXY_GPG_KEYRING=~/.ansible/pubring.kbx
export ANSIBLE_GALAXY_REQUIRED_VALID_SIGNATURE_COUNT=all
ansible-galaxy collection install my_namespace.my_collection --ignore-signature-status-
↪code NO_PUBKEY
```

If verification fails for the example above, only errors other than NO_PUBKEY will be displayed.

If verification is unsuccessful, the collection will not be installed. GnuPG signature verification can be disabled with `--disable-gpg-verify` or by configuring *GALAXY_DISABLE_GPG_VERIFY*.

Installing an older version of a collection

You can only have one version of a collection installed at a time. By default `ansible-galaxy` installs the latest available version. If you want to install a specific version, you can add a version range identifier. For example, to install the 1.0.0-beta.1 version of the collection:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.0-beta.1
```

You can specify multiple range identifiers separated by `,`. Use single quotes so the shell passes the entire command, including `>`, `!`, and other operators, along. For example, to install the most recent version that is greater than or equal to 1.0.0 and less than 2.0.0:

```
ansible-galaxy collection install 'my_namespace.my_collection:>=1.0.0,<2.0.0'
```

Ansible will always install the most recent version that meets the range identifiers you specify. You can use the following range identifiers:

- `*`: The most recent version. This is the default.
- `!=`: Not equal to the version specified.
- `==`: Exactly the version specified.
- `>=`: Greater than or equal to the version specified.
- `>`: Greater than the version specified.
- `<=`: Less than or equal to the version specified.
- `<`: Less than the version specified.

Note: By default `ansible-galaxy` ignores pre-release versions. To install a pre-release version, you must use the == range identifier to require it explicitly.

Install multiple collections with a requirements file

You can set up a `requirements.yml` file to install multiple collections in one command. This file is a YAML file in the format:

```
---
collections:
# With just the collection name
- my_namespace.my_collection

# With the collection name, version, and source options
- name: my_namespace.my_other_collection
  version: ">=1.2.0" # Version range identifiers (default: ``*``)
  source: ... # The Galaxy URL to pull the collection from (default: ``--api-server``␣
↪from cmdline)
```

You can specify the following keys for each collection entry:

- `name`
- `version`
- `signatures`
- `source`
- `type`

The `version` key uses the same range identifier format documented in *Installing an older version of a collection*.

The `signatures` key accepts a list of signature sources that are used to supplement those found on the Galaxy server during collection installation and `ansible-galaxy collection verify`. Signature sources should be URIs that contain the detached signature. The `--keyring` CLI option must be provided if signatures are specified.

Signatures are only used to verify collections on Galaxy servers. User-provided signatures are not used to verify collections installed from git repositories, source directories, or URLs/paths to tar.gz files.

```
collections:
  - name: namespace.name
    version: 1.0.0
    type: galaxy
    signatures:
      - https://examplehost.com/detached_signature.asc
      - file:///path/to/local/detached_signature.asc
```

The `type` key can be set to `file`, `galaxy`, `git`, `url`, `dir`, or `subdirs`. If `type` is omitted, the `name` key is used to implicitly determine the source of the collection.

When you install a collection with `type: git`, the `version` key can refer to a branch or to a git commit-ish object (commit or tag). For example:

```yaml
collections:
  - name: https://github.com/organization/repo_name.git
    type: git
    version: devel
```

You can also add roles to a `requirements.yml` file, under the `roles` key. The values follow the same format as a requirements file used in older Ansible releases.

```yaml
---
roles:
  # Install a role from Ansible Galaxy.
  - name: geerlingguy.java
    version: "1.9.6" # note that ranges are not supported for roles


collections:
  # Install a collection from Ansible Galaxy.
  - name: geerlingguy.php_roles
    version: ">=0.9.3"
    source: https://galaxy.ansible.com
```

To install both roles and collections at the same time with one command, run the following:

```
$ ansible-galaxy install -r requirements.yml
```

Running `ansible-galaxy collection install -r` or `ansible-galaxy role install -r` will only install collections, or roles respectively.

Note: Installing both roles and collections from the same requirements file will not work when specifying a custom collection or role install path. In this scenario the collections will be skipped and the command will process each like `ansible-galaxy role install` would.

Downloading a collection for offline use

To download the collection tarball from Galaxy for offline use:

1. Navigate to the collection page.
2. Click on *Download tarball*.

You may also need to manually download any dependent collections.

Installing a collection from source files

Ansible can also install from a source directory in several ways:

```yaml
collections:
  # directory containing the collection
  - source: ./my_namespace/my_collection/
    type: dir

  # directory containing a namespace, with collections as subdirectories
```

(continues on next page)

(continued from previous page)

```
  - source: ./my_namespace/
    type: subdirs
```

Ansible can also install a collection collected with `ansible-galaxy collection build` or downloaded from Galaxy for offline use by specifying the output file directly:

```
collections:
  - name: /tmp/my_namespace-my_collection-1.0.0.tar.gz
    type: file
```

Note: Relative paths are calculated from the current working directory (where you are invoking `ansible-galaxy install -r` from). They are not taken relative to the `requirements.yml` file.

Installing a collection from a git repository

You can install a collection from a git repository instead of from Galaxy or Automation Hub. As a developer, installing from a git repository lets you review your collection before you create the tarball and publish the collection. As a user, installing from a git repository lets you use collections or versions that are not in Galaxy or Automation Hub yet.

The repository must contain a `galaxy.yml` or `MANIFEST.json` file. This file provides metadata such as the version number and namespace of the collection.

Installing a collection from a git repository at the command line

To install a collection from a git repository at the command line, use the URI of the repository instead of a collection name or path to a `tar.gz` file. Use the prefix `git+`, unless you're using SSH authentication with the user `git` (for example, `git@github.com:ansible-collections/ansible.windows.git`). You can specify a branch, commit, or tag using the comma-separated git commit-ish syntax.

For example:

```
# Install a collection in a repository using the latest commit on the branch 'devel'
ansible-galaxy collection install git+https://github.com/organization/repo_name.git,devel

# Install a collection from a private github repository
ansible-galaxy collection install git@github.com:organization/repo_name.git

# Install a collection from a local git repository
ansible-galaxy collection install git+file:///home/user/path/to/repo_name.git
```

Warning: Embedding credentials into a git URI is not secure. Use safe authentication options to prevent your credentials from being exposed in logs or elsewhere.

- Use SSH authentication
- Use netrc authentication
- Use http.extraHeader in your git configuration
- Use url.<base>.pushInsteadOf in your git configuration

Specifying the collection location within the git repository

When you install a collection from a git repository, Ansible uses the collection `galaxy.yml` or `MANIFEST.json` metadata file to build the collection. By default, Ansible searches two paths for collection `galaxy.yml` or `MANIFEST.json` metadata files:

- The top level of the repository.
- Each directory in the repository path (one level deep).

If a `galaxy.yml` or `MANIFEST.json` file exists in the top level of the repository, Ansible uses the collection metadata in that file to install an individual collection.

```
├── galaxy.yml
├── plugins/
│   ├── lookup/
│   ├── modules/
│   └── module_utils/
└─── README.md
```

If a `galaxy.yml` or `MANIFEST.json` file exists in one or more directories in the repository path (one level deep), Ansible installs each directory with a metadata file as a collection. For example, Ansible installs both collection1 and collection2 from this repository structure by default:

```
├── collection1
│   ├── docs/
│   ├── galaxy.yml
│   └── plugins/
│       ├── inventory/
│       └── modules/
└── collection2
    ├── docs/
    ├── galaxy.yml
    ├── plugins/
    |   ├── filter/
    |   └── modules/
    └── roles/
```

If you have a different repository structure or only want to install a subset of collections, you can add a fragment to the end of your URI (before the optional comma-separated version) to indicate the location of the metadata file or files. The path should be a directory, not the metadata file itself. For example, to install only collection2 from the example repository with two collections:

```
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪collection2/
```

In some repositories, the main directory corresponds to the namespace:

```
namespace/
├── collectionA/
|   ├── docs/
|   ├── galaxy.yml
|   ├── plugins/
|   │   ├── README.md
|   │   └── modules/
```

(continues on next page)

(continued from previous page)

```
|   ├── README.md
|   └── roles/
└── collectionB/
    ├── docs/
    ├── galaxy.yml
    ├── plugins/
    │   ├── connection/
    │   └── modules/
    ├── README.md
    └── roles/
```

You can install all collections in this repository, or install one collection from a specific commit:

```
# Install all collections in the namespace
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪namespace/

# Install an individual collection using a specific commit
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪namespace/collectionA/,7b60ddc245bc416b72d8ea6ed7b799885110f5e5
```

Configuring the `ansible-galaxy` client

By default, `ansible-galaxy` uses https://galaxy.ansible.com as the Galaxy server (as listed in the `ansible.cfg` file under *GALAXY_SERVER*).

You can use either option below to configure `ansible-galaxy collection` to use other servers (such as a custom Galaxy server):

- Set the server list in the *GALAXY_SERVER_LIST* configuration option in *The configuration file*.
- Use the `--server` command line argument to limit to an individual server.

To configure a Galaxy server list in `ansible.cfg`:

1. Add the `server_list` option under the `[galaxy]` section to one or more server names.
2. Create a new section for each server name.
3. Set the `url` option for each server name.
4. Optionally, set the API token for each server name. Go to https://galaxy.ansible.com/me/preferences and click *Show API key*.

Note: The `url` option for each server name must end with a forward slash `/`. If you do not set the API token in your Galaxy server list, use the `--api-key` argument to pass in the token to the `ansible-galaxy collection publish` command.

The following example shows how to configure multiple servers:

```
[galaxy]
server_list = my_org_hub, release_galaxy, test_galaxy, my_galaxy_ng

[galaxy_server.my_org_hub]
```

(continues on next page)

(continued from previous page)

```
url=https://automation.my_org/
username=my_user
password=my_pass

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=my_token

[galaxy_server.test_galaxy]
url=https://galaxy-dev.ansible.com/
token=my_test_token

[galaxy_server.my_galaxy_ng]
url=http://my_galaxy_ng:8000/api/automation-hub/
auth_url=http://my_keycloak:8080/auth/realms/myco/protocol/openid-connect/token
client_id=galaxy-ng
token=my_keycloak_access_token
```

Note: You can use the `--server` command line argument to select an explicit Galaxy server in the `server_list` and the value of this argument should match the name of the server. To use a server not in the server list, set the value to the URL to access that server (all servers in the server list will be ignored). Also you cannot use the `--api-key` argument for any of the predefined servers. You can only use the `api_key` argument if you did not define a server list or if you specify a URL in the `--server` argument.

Galaxy server list configuration options

The *GALAXY_SERVER_LIST* option is a list of server identifiers in a prioritized order. When searching for a collection, the install process will search in that order, for example, `automation_hub` first, then `my_org_hub`, `release_galaxy`, and finally `test_galaxy` until the collection is found. The actual Galaxy instance is then defined under the section `[galaxy_server.{{ id }}]` where `{{ id }}` is the server identifier defined in the list. This section can then define the following keys:

- `url`: The URL of the Galaxy instance to connect to. Required.
- `token`: An API token key to use for authentication against the Galaxy instance. Mutually exclusive with `username`.
- `username`: The username to use for basic authentication against the Galaxy instance. Mutually exclusive with `token`.
- `password`: The password to use, in conjunction with `username`, for basic authentication.
- `auth_url`: The URL of a Keycloak server 'token_endpoint' if using SSO authentication (for example, galaxyNG). Mutually exclusive with `username`. Requires `token`.
- `validate_certs`: Whether or not to verify TLS certificates for the Galaxy server. This defaults to True unless the `--ignore-certs` option is provided or `GALAXY_IGNORE_CERTS` is configured to True.
- `client_id`: The Keycloak token's client_id to use for authentication. Requires `auth_url` and `token`. The default `client_id` is cloud-services to work with Red Hat SSO.

As well as defining these server options in the `ansible.cfg` file, you can also define them as environment variables. The environment variable is in the form `ANSIBLE_GALAXY_SERVER_{{ id }}_{{ key }}` where `{{ id }}` is the upper case form of the server identifier and `{{ key }}` is the key to define. For example I can define `token` for `release_galaxy` by setting `ANSIBLE_GALAXY_SERVER_RELEASE_GALAXY_TOKEN=secret_token`.

For operations that use only one Galaxy server (for example, the `publish`, `info`, or `install` commands). the `ansible-galaxy collection` command uses the first entry in the `server_list`, unless you pass in an explicit server with the `--server` argument.

Note: `ansible-galaxy` can seek out dependencies on other configured Galaxy instances to support the use case where a collection can depend on a collection from another Galaxy instance.

1.9.2 Downloading collections

To download a collection and its dependencies for an offline install, run `ansible-galaxy collection download`. This downloads the collections specified and their dependencies to the specified folder and creates a `requirements.yml` file which can be used to install those collections on a host without access to a Galaxy server. All the collections are downloaded by default to the `./collections` folder.

Just like the `install` command, the collections are sourced based on the *configured galaxy server config*. Even if a collection to download was specified by a URL or path to a tarball, the collection will be redownloaded from the configured Galaxy server.

Collections can be specified as one or multiple collections or with a `requirements.yml` file just like `ansible-galaxy collection install`.

To download a single collection and its dependencies:

```
ansible-galaxy collection download my_namespace.my_collection
```

To download a single collection at a specific version:

```
ansible-galaxy collection download my_namespace.my_collection:1.0.0
```

To download multiple collections either specify multiple collections as command line arguments as shown above or use a requirements file in the format documented with *Install multiple collections with a requirements file*.

```
ansible-galaxy collection download -r requirements.yml
```

You can also download a source collection directory. The collection is built with the mandatory `galaxy.yml` file.

```
ansible-galaxy collection download /path/to/collection

ansible-galaxy collection download git+file:///path/to/collection/.git
```

You can download multiple source collections from a single namespace by providing the path to the namespace.

```
ns/
├── collection1/
│   ├── galaxy.yml
│   └── plugins/
└── collection2/
    ├── galaxy.yml
    └── plugins/
```

```
ansible-galaxy collection install /path/to/ns
```

All the collections are downloaded by default to the `./collections` folder but you can use `-p` or `--download-path` to specify another path:

```
ansible-galaxy collection download my_namespace.my_collection -p ~/offline-collections
```

Once you have downloaded the collections, the folder contains the collections specified, their dependencies, and a `requirements.yml` file. You can use this folder as is with `ansible-galaxy collection install` to install the collections on a host without access to a Galaxy server.

```
# This must be run from the folder that contains the offline collections and␣
↪requirements.yml file downloaded
# by the internet-connected host
cd ~/offline-collections
ansible-galaxy collection install -r requirements.yml
```

1.9.3 Listing collections

To list installed collections, run `ansible-galaxy collection list`. This shows all of the installed collections found in the configured collections search paths. It will also show collections under development which contain a galaxy.yml file instead of a MANIFEST.json. The path where the collections are located are displayed as well as version information. If no version information is available, a * is displayed for the version number.

```
# /home/astark/.ansible/collections/ansible_collections
Collection                 Version
-------------------------- -------
cisco.aci                  0.0.5
cisco.mso                  0.0.4
sandwiches.ham             *
splunk.es                  0.0.5

# /usr/share/ansible/collections/ansible_collections
Collection        Version
----------------- -------
fortinet.fortios  1.0.6
pureport.pureport 0.0.8
sensu.sensu_go    1.3.0
```

Run with -vvv to display more detailed information. You may see additional collections here that were added as dependencies of your installed collections. Only use collections in your playbooks that you have directly installed.

To list a specific collection, pass a valid fully qualified collection name (FQCN) to the command `ansible-galaxy collection list`. All instances of the collection will be listed.

```
> ansible-galaxy collection list fortinet.fortios

# /home/astark/.ansible/collections/ansible_collections
Collection       Version
---------------- -------
fortinet.fortios 1.0.1

# /usr/share/ansible/collections/ansible_collections
Collection       Version
---------------- -------
fortinet.fortios 1.0.6
```

To search other paths for collections, use the -p option. Specify multiple search paths by separating them with a :. The list of paths specified on the command line will be added to the beginning of the configured collections search paths.

```
> ansible-galaxy collection list -p '/opt/ansible/collections:/etc/ansible/collections'

# /opt/ansible/collections/ansible_collections
Collection      Version
--------------- -------
sandwiches.club 1.7.2

# /etc/ansible/collections/ansible_collections
Collection     Version
-------------- -------
sandwiches.pbj 1.2.0

# /home/astark/.ansible/collections/ansible_collections
Collection                 Version
-------------------------- -------
cisco.aci                  0.0.5
cisco.mso                  0.0.4
fortinet.fortios           1.0.1
sandwiches.ham             *
splunk.es                  0.0.5

# /usr/share/ansible/collections/ansible_collections
Collection        Version
----------------- -------
fortinet.fortios  1.0.6
pureport.pureport 0.0.8
sensu.sensu_go    1.3.0
```

1.9.4 Verifying collections

Verifying collections with ansible-galaxy

Once installed, you can verify that the content of the installed collection matches the content of the collection on the server. This feature expects that the collection is installed in one of the configured collection paths and that the collection exists on one of the configured galaxy servers.

```
ansible-galaxy collection verify my_namespace.my_collection
```

The output of the `ansible-galaxy collection verify` command is quiet if it is successful. If a collection has been modified, the altered files are listed under the collection name.

```
ansible-galaxy collection verify my_namespace.my_collection
Collection my_namespace.my_collection contains modified content in the following files:
my_namespace.my_collection
    plugins/inventory/my_inventory.py
    plugins/modules/my_module.py
```

You can use the -vvv flag to display additional information, such as the version and path of the installed collection, the URL of the remote collection used for validation, and successful verification output.

```
ansible-galaxy collection verify my_namespace.my_collection -vvv
...
Verifying 'my_namespace.my_collection:1.0.0'.
Installed collection found at '/path/to/ansible_collections/my_namespace/my_collection/'
Remote collection found at 'https://galaxy.ansible.com/download/my_namespace-my_
↪collection-1.0.0.tar.gz'
Successfully verified that checksums for 'my_namespace.my_collection:1.0.0' match the␣
↪remote collection
```

If you have a pre-release or non-latest version of a collection installed you should include the specific version to verify. If the version is omitted, the installed collection is verified against the latest version available on the server.

```
ansible-galaxy collection verify my_namespace.my_collection:1.0.0
```

In addition to the `namespace.collection_name:version` format, you can provide the collections to verify in a `requirements.yml` file. Dependencies listed in `requirements.yml` are not included in the verify process and should be verified separately.

```
ansible-galaxy collection verify -r requirements.yml
```

Verifying against `tar.gz` files is not supported. If your `requirements.yml` contains paths to tar files or URLs for installation, you can use the `--ignore-errors` flag to ensure that all collections using the `namespace.name` format in the file are processed.

Verifying signed collections

If a collection has been signed by a *distribution server*, the server will provide ASCII armored, detached signatures to verify the authenticity of the MANIFEST.json before using it to verify the collection's contents. This option is not available on all distribution servers. See *Distributing collections* for a table listing which servers support collection signing. See *Installing collections with signature verification* for how to verify a signed collection when you install it.

To verify a signed installed collection:

```
ansible-galaxy collection verify my_namespace.my_collection  --keyring ~/.ansible/
↪pubring.kbx
```

Use the `--signature` option to verify collection name(s) provided on the CLI with an additional signature. This option can be used multiple times to provide multiple signatures.

```
ansible-galaxy collection verify my_namespace.my_collection --signature https://
↪examplehost.com/detached_signature.asc --signature file:///path/to/local/detached_
↪signature.asc --keyring ~/.ansible/pubring.kbx
```

Optionally, you can verify a collection signature with a `requirements.yml` file.

```
ansible-galaxy collection verify -r requirements.yml --keyring ~/.ansible/pubring.kbx
```

When a collection is installed from a distribution server, the signatures provided by the server to verify the collection's authenticity are saved alongside the installed collections. This data is used to verify the internal consistency of the collection without querying the distribution server again when the `--offline` option is provided.

```
ansible-galaxy collection verify my_namespace.my_collection --offline --keyring ~/.
↪ansible/pubring.kbx
```

1.9.5 Using collections in a playbook

Once installed, you can reference a collection content by its fully qualified collection name (FQCN):

```
- hosts: all
  tasks:
    - my_namespace.my_collection.mymodule:
        option1: value
```

This works for roles or any type of plugin distributed within the collection:

```
- hosts: all
  tasks:
    - import_role:
        name: my_namespace.my_collection.role1

    - my_namespace.mycollection.mymodule:
        option1: value

    - debug:
        msg: '{{ lookup("my_namespace.my_collection.lookup1", 'param1')| my_namespace.my_
↪collection.filter1 }}'
```

Simplifying module names with the `collections` keyword

The `collections` keyword lets you define a list of collections that your role or playbook should search for unqualified module and action names. So you can use the `collections` keyword, then simply refer to modules and action plugins by their short-form names throughout that role or playbook.

> **Warning:** If your playbook uses both the `collections` keyword and one or more roles, the roles do not inherit the collections set by the playbook. This is one of the reasons we recommend you always use FQCN. See below for roles details.

Using `collections` in roles

Within a role, you can control which collections Ansible searches for the tasks inside the role using the `collections` keyword in the role's `meta/main.yml`. Ansible will use the collections list defined inside the role even if the playbook that calls the role defines different collections in a separate `collections` keyword entry. Roles defined inside a collection always implicitly search their own collection first, so you don't need to use the `collections` keyword to access modules, actions, or other roles contained in the same collection.

```
# myrole/meta/main.yml
collections:
  - my_namespace.first_collection
  - my_namespace.second_collection
  - other_namespace.other_collection
```

Using `collections` in playbooks

In a playbook, you can control the collections Ansible searches for modules and action plugins to execute. However, any roles you call in your playbook define their own collections search order; they do not inherit the calling playbook's settings. This is true even if the role does not define its own `collections` keyword.

```
- hosts: all
  collections:
    - my_namespace.my_collection

  tasks:
    - import_role:
        name: role1

    - mymodule:
        option1: value

    - debug:
        msg: '{{ lookup("my_namespace.my_collection.lookup1", "param1")| my_namespace.my_
↪collection.filter1 }}'
```

The `collections` keyword merely creates an ordered 'search path' for non-namespaced plugin and role references. It does not install content or otherwise change Ansible's behavior around the loading of plugins or roles. Note that an FQCN is still required for non-action or module plugins (for example, lookups, filters, tests).

When using the `collections` keyword, it is not necessary to add in `ansible.builtin` as part of the search list. When left omitted, the following content is available by default:

1. Standard ansible modules and plugins available through `ansible-base/ansible-core`
2. Support for older 3rd party plugin paths

In general, it is preferable to use a module or plugin's FQCN over the `collections` keyword and the short name for all content in `ansible-core`

Using a playbook from a collection

New in version 2.11.

You can also distribute playbooks in your collection and invoke them using the same semantics you use for plugins:

```
ansible-playbook my_namespace.my_collection.playbook1 -i ./myinventory
```

From inside a playbook:

```
- import_playbook: my_namespace.my_collection.playbookX
```

A few recommendations when creating such playbooks, `hosts:` should be generic or at least have a variable input.

```
- hosts: all  # Use --limit or customized inventory to restrict hosts targeted

- hosts: localhost  # For things you want to restrict to the controller

- hosts: '{{target|default("webservers")}}'  # Assumes inventory provides a 'webservers'␣
↪group, but can also use ``-e 'target=host1,host2'``
```

This will have an implied entry in the `collections:` keyword of `my_namespace.my_collection` just as with roles.

Note:

- Playbook names, like other collection resources, have a restricted set of valid characters. Names can contain only lowercase alphanumeric characters, plus _ and must start with an alpha character. The dash - character is not valid for playbook names in collections. Playbooks whose names contain invalid characters are not addressable: this is a limitation of the Python importer that is used to load collection resources.
- Playbooks in collections do not support 'adjacent' plugins, all plugins must be in the collection specific directories.

1.9.6 Collections index

You can find an index of collections at Collection Index.

1.10 Using Ansible on Windows and BSD

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible guide for Microsoft Windows and BSD. Because Windows is not a POSIX-compliant operating system, Ansible interacts with Windows hosts differently to Linux/Unix hosts. Likewise managing hosts that run BSD is different to managing other Unix-like host operating systems. Find out everything you need to know about using Ansible on Windows and with BSD hosts.

1.10.1 Setting up a Windows Host

This document discusses the setup that is required before Ansible can communicate with a Microsoft Windows host.

Host Requirements

For Ansible to communicate to a Windows host and use Windows modules, the Windows host must meet these base requirements for connectivity:

- With Ansible you can generally manage Windows versions under the current and extended support from Microsoft. You can also manage desktop OSs including Windows 8.1, and 10, and server OSs including Windows Server 2012, 2012 R2, 2016, 2019, and 2022.
- You need to install PowerShell 3.0 or newer and at least .NET 4.0 on the Windows host.
- You need to create and activate a WinRM listener. More details, see WinRM Setup.

Note: Some Ansible modules have additional requirements, such as a newer OS or PowerShell version. Consult the module documentation page to determine whether a host meets those requirements.

Upgrading PowerShell and .NET Framework

Ansible requires PowerShell version 3.0 and .NET Framework 4.0 or newer to function on older operating systems like Server 2008 and Windows 7. The base image does not meet this requirement. You can use the Upgrade-PowerShell.ps1 script to update these.

This is an example of how to run this script from PowerShell:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
$url = "https://raw.githubusercontent.com/jborean93/ansible-windows/master/scripts/
↪Upgrade-PowerShell.ps1"
$file = "$env:temp\Upgrade-PowerShell.ps1"
$username = "Administrator"
$password = "Password"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Force

&$file -Version 5.1 -Username $username -Password $password -Verbose
```

In the script, the `file` value can be the PowerShell version 3.0, 4.0, or 5.1.

Once completed, you need to run the following PowerShell commands:

1. As an optional but good security practice, you can set the execution policy back to the default.

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Force
```

Use the `RemoteSigned` value for Windows servers, or `Restricted` for Windows clients.

2. Remove the auto logon.

```
$reg_winlogon_path = "HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
Set-ItemProperty -Path $reg_winlogon_path -Name AutoAdminLogon -Value 0
Remove-ItemProperty -Path $reg_winlogon_path -Name DefaultUserName -ErrorAction␣
↪SilentlyContinue
Remove-ItemProperty -Path $reg_winlogon_path -Name DefaultPassword -ErrorAction␣
↪SilentlyContinue
```

The script determines what programs you need to install (such as .NET Framework 4.5.2) and what PowerShell version needs to be present. If a reboot is needed and the `username` and `password` parameters are set, the script will automatically reboot the machine and then logon. If the `username` and `password` parameters are not set, the script will prompt the user to manually reboot and logon when required. When the user is next logged in, the script will continue where it left off and the process continues until no more actions are required.

Note: If you run the script on Server 2008, then you need to install SP2. For Server 2008 R2 or Windows 7 you need SP1.

On Windows Server 2008 you can install only PowerShell 3.0. A newer version will result in the script failure.

The `username` and `password` parameters are stored in plain text in the registry. Run the cleanup commands after the script finishes to ensure no credentials are stored on the host.

WinRM Memory Hotfix

On PowerShell v3.0, there is a bug that limits the amount of memory available to the WinRM service. Use the Install-WMF3Hotfix.ps1 script to install a hotfix on affected hosts as part of the system bootstrapping or imaging process. Without this hotfix, Ansible fails to execute certain commands on the Windows host.

To install the hotfix:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
$url = "https://raw.githubusercontent.com/jborean93/ansible-windows/master/scripts/
↪Install-WMF3Hotfix.ps1"
$file = "$env:temp\Install-WMF3Hotfix.ps1"

(New-Object -TypeName System.Net.WebClient).DownloadFile($url, $file)
powershell.exe -ExecutionPolicy ByPass -File $file -Verbose
```

For more details, refer to the "Out of memory" error on a computer that has a customized MaxMemoryPerShellMB quota set and has WMF 3.0 installed article.

WinRM Setup

You need to configure the WinRM service so that Ansible can connect to it. There are two main components of the WinRM service that governs how Ansible can interface with the Windows host: the `listener` and the `service` configuration settings.

WinRM Listener

The WinRM services listen for requests on one or more ports. Each of these ports must have a listener created and configured.

To view the current listeners that are running on the WinRM service:

```
winrm enumerate winrm/config/Listener
```

This will output something like:

```
Listener
    Address = *
    Transport = HTTP
    Port = 5985
    Hostname
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint
    ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1, fe80::5efe:10.0.2.15%6,␣
↪fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3, fe80::e8ea:d765:2c69:7756%7

Listener
    Address = *
    Transport = HTTPS
    Port = 5986
    Hostname = SERVER2016
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint = E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE
    ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1, fe80::5efe:10.0.2.15%6,␣
↪fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3, fe80::e8ea:d765:2c69:7756%7
```

In the example above there are two listeners activated. One is listening on port 5985 over HTTP and the other is listening on port 5986 over HTTPS. Some of the key options that are useful to understand are:

- `Transport`: Whether the listener is run over HTTP or HTTPS. We recommend you use a listener over HTTPS because the data is encrypted without any further changes required.
- `Port`: The port the listener runs on. By default it is `5985` for HTTP and `5986` for HTTPS. This port can be changed to whatever is required and corresponds to the host var `ansible_port`.
- `URLPrefix`: The URL prefix to listen on. By default it is `wsman`. If you change this option, you need to set the host var `ansible_winrm_path` to the same value.
- `CertificateThumbprint`: If you use an HTTPS listener, this is the thumbprint of the certificate in the Windows Certificate Store that is used in the connection. To get the details of the certificate itself, run this command with the relevant certificate thumbprint in PowerShell:

```
$thumbprint = "E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE"
Get-ChildItem -Path cert:\LocalMachine\My -Recurse | Where-Object { $_.Thumbprint -eq
↪$thumbprint } | Select-Object *
```

Setup WinRM Listener

There are three ways to set up a WinRM listener:

- Using `winrm quickconfig` for HTTP or `winrm quickconfig -transport:https` for HTTPS. This is the easiest option to use when running outside of a domain environment and a simple listener is required. Unlike the other options, this process also has the added benefit of opening up the firewall for the ports required and starts the WinRM service.
- Using Group Policy Objects (GPO). This is the best way to create a listener when the host is a member of a domain because the configuration is done automatically without any user input. For more information on group policy objects, see the Group Policy Objects documentation.
- Using PowerShell to create a listener with a specific configuration. This can be done by running the following PowerShell commands:

  ```
  $selector_set = @{
      Address = "*"
      Transport = "HTTPS"
  }
  $value_set = @{
      CertificateThumbprint = "E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE"
  }

  New-WSManInstance -ResourceURI "winrm/config/Listener" -SelectorSet $selector_set -
  ↪ValueSet $value_set
  ```

  To see the other options with this PowerShell command, refer to the New-WSManInstance documentation.

Note: When creating an HTTPS listener, you must create and store a certificate in the `LocalMachine\My` certificate store.

Delete WinRM Listener

- To remove all WinRM listeners:

```
Remove-Item -Path WSMan:\localhost\Listener\* -Recurse -Force
```

- To remove only those listeners that run over HTTPS:

```
Get-ChildItem -Path WSMan:\localhost\Listener | Where-Object { $_.Keys -contains
↪"Transport=HTTPS" } | Remove-Item -Recurse -Force
```

Note: The `Keys` object is an array of strings, so it can contain different values. By default, it contains a key for `Transport=` and `Address=` which correspond to the values from the `winrm enumerate winrm/config/Listeners` command.

WinRM Service Options

You can control the behavior of the WinRM service component, including authentication options and memory settings.

To get an output of the current service configuration options, run the following command:

```
winrm get winrm/config/Service
winrm get winrm/config/Winrs
```

This will output something like:

```
Service
    RootSDDL = O:NSG:BAD:P(A;;GA;;;BA)(A;;GR;;;IU)S:P(AU;FA;GA;;;WD)(AU;SA;GXGW;;;WD)
    MaxConcurrentOperations = 4294967295
    MaxConcurrentOperationsPerUser = 1500
    EnumerationTimeoutms = 240000
    MaxConnections = 300
    MaxPacketRetrievalTimeSeconds = 120
    AllowUnencrypted = false
    Auth
        Basic = true
        Kerberos = true
        Negotiate = true
        Certificate = true
        CredSSP = true
        CbtHardeningLevel = Relaxed
    DefaultPorts
        HTTP = 5985
        HTTPS = 5986
    IPv4Filter = *
    IPv6Filter = *
    EnableCompatibilityHttpListener = false
    EnableCompatibilityHttpsListener = false
    CertificateThumbprint
    AllowRemoteAccess = true

Winrs
    AllowRemoteShellAccess = true
    IdleTimeout = 7200000
    MaxConcurrentUsers = 2147483647
    MaxShellRunTime = 2147483647
    MaxProcessesPerShell = 2147483647
    MaxMemoryPerShellMB = 2147483647
    MaxShellsPerUser = 2147483647
```

You do not need to change the majority of these options. However, some of the important ones to know about are:

- `Service\AllowUnencrypted` - specifies whether WinRM will allow HTTP traffic without message encryption. Message level encryption is only possible when the `ansible_winrm_transport` variable is `ntlm`, `kerberos` or `credssp`. By default, this is `false` and you should only set it to `true` when debugging WinRM messages.
- `Service\Auth\*` - defines what authentication options you can use with the WinRM service. By default, `Negotiate (NTLM)` and `Kerberos` are enabled.
- `Service\Auth\CbtHardeningLevel` - specifies whether channel binding tokens are not verified (None), verified but not required (Relaxed), or verified and required (Strict). CBT is only used when connecting with NT

LAN Manager (NTLM) or Kerberos over HTTPS.

- Service\CertificateThumbprint - thumbprint of the certificate for encrypting the TLS channel used with CredSSP authentication. By default, this is empty. A self-signed certificate is generated when the WinRM service starts and is used in the TLS process.
- Winrs\MaxShellRunTime - maximum time, in milliseconds, that a remote command is allowed to execute.
- Winrs\MaxMemoryPerShellMB - maximum amount of memory allocated per shell, including its child processes.

To modify a setting under the Service key in PowerShell, you need to provide a path to the option after winrm/config/Service:

```
Set-Item -Path WSMan:\localhost\Service\{path} -Value {some_value}
```

For example, to change Service\Auth\CbtHardeningLevel:

```
Set-Item -Path WSMan:\localhost\Service\Auth\CbtHardeningLevel -Value Strict
```

To modify a setting under the Winrs key in PowerShell, you need to provide a path to the option after winrm/config/Winrs:

```
Set-Item -Path WSMan:\localhost\Shell\{path} -Value {some_value}
```

For example, to change Winrs\MaxShellRunTime:

```
Set-Item -Path WSMan:\localhost\Shell\MaxShellRunTime -Value 2147483647
```

Note: If you run the command in a domain environment, some of these options are set by GPO and cannot be changed on the host itself. When you configured a key with GPO, it contains the text [Source="GPO"] next to the value.

Common WinRM Issues

WinRM has a wide range of configuration options, which makes its configuration complex. As a result, errors that Ansible displays could in fact be problems with the host setup instead.

To identify a host issue, run the following command from another Windows host to connect to the target Windows host.

- To test HTTP:

```
winrs -r:http://server:5985/wsman -u:Username -p:Password ipconfig
```

- To test HTTPS:

```
winrs -r:https://server:5986/wsman -u:Username -p:Password -ssl ipconfig
```

The command will fail if the certificate is not verifiable.

- To test HTTPS ignoring certificate verification:

```
$username = "Username"
$password = ConvertTo-SecureString -String "Password" -AsPlainText -Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
↪$username, $password
```

(continues on next page)

(continued from previous page)

```
$session_option = New-PSSessionOption -SkipCACheck -SkipCNCheck -SkipRevocationCheck
Invoke-Command -ComputerName server -UseSSL -ScriptBlock { ipconfig } -Credential $cred -
↪SessionOption $session_option
```

If any of the above commands fail, the issue is probably related to the WinRM setup.

HTTP 401/Credentials Rejected

An HTTP 401 error indicates the authentication process failed during the initial connection. You can check the following to troubleshoot:

- The credentials are correct and set properly in your inventory with the `ansible_user` and `ansible_password` variables.
- The user is a member of the local Administrators group, or has been explicitly granted access. You can perform a connection test with the `winrs` command to rule this out.
- The authentication option set by the `ansible_winrm_transport` variable is enabled under `Service\Auth\*`.
- If running over HTTP and not HTTPS, use `ntlm`, `kerberos` or `credssp` with the `ansible_winrm_message_encryption: auto` custom inventory variable to enable message encryption. If you use another authentication option, or if it is not possible to upgrade the installed `pywinrm` package, you can set `Service\AllowUnencrypted` to `true`. This is recommended only for troubleshooting.
- The downstream packages `pywinrm`, `requests-ntlm`, `requests-kerberos`, and/or `requests-credssp` are up to date using `pip`.
- For Kerberos authentication, ensure that `Service\Auth\CbtHardeningLevel` is not set to `Strict`.
- For Basic or Certificate authentication, make sure that the user is a local account. Domain accounts do not work with Basic and Certificate authentication.

HTTP 500 Error

An HTTP 500 error indicates a problem with the WinRM service. You can check the following to troubleshoot:

- The number of your currently open shells has not exceeded either `WinRsMaxShellsPerUser`. Alternatively, you did not exceed any of the other Winrs quotas.

Timeout Errors

Sometimes Ansible is unable to reach the host. These instances usually indicate a problem with the network connection. You can check the following to troubleshoot:

- The firewall is not set to block the configured WinRM listener ports.
- A WinRM listener is enabled on the port and path set by the host vars.
- The `winrm` service is running on the Windows host and is configured for the automatic start.

Connection Refused Errors

When you communicate with the WinRM service on the host you can encounter some problems. Check the following to help the troubleshooting:

- The WinRM service is up and running on the host. Use the `(Get-Service -Name winrm).Status` command to get the status of the service.
- The host firewall is allowing traffic over the WinRM port. By default this is `5985` for HTTP and `5986` for HTTPS.

Sometimes an installer may restart the WinRM or HTTP service and cause this error. The best way to deal with this is to use the `win_psexec` module from another Windows host.

Failure to Load Builtin Modules

Sometimes PowerShell fails with an error message similar to:

```
The 'Out-String' command was found in the module 'Microsoft.PowerShell.Utility', but the␣
↪module could not be loaded.
```

In that case, there could be a problem when trying to access all the paths specified by the `PSModulePath` environment variable.

A common cause of this issue is that `PSModulePath` contains a Universal Naming Convention (UNC) path to a file share. Additionally, the double hop/credential delegation issue causes that the Ansible process cannot access these folders. To work around this problem is to either:

- Remove the UNC path from `PSModulePath`.

or

- Use an authentication option that supports credential delegation like `credssp` or `kerberos`. You need to have the credential delegation enabled.

See KB4076842 for more information on this problem.

Windows SSH Setup

Ansible 2.8 has added an experimental SSH connection for Windows managed nodes.

Warning: Use this feature at your own risk! Using SSH with Windows is experimental. This implementation may make backwards incompatible changes in future releases. The server-side components can be unreliable depending on your installed version.

Installing OpenSSH using Windows Settings

You can use OpenSSH to connect Window 10 clients to Windows Server 2019. OpenSSH Client is available to install on Windows 10 build 1809 and later. OpenSSH Server is available to install on Windows Server 2019 and later.

For more information, refer to Get started with OpenSSH for Windows.

Installing Win32-OpenSSH

To install the Win32-OpenSSH service for use with Ansible, select one of these installation options:

- Manually install `Win32-OpenSSH`, following the install instructions from Microsoft.
- Use Chocolatey:

```
choco install --package-parameters=/SSHServerFeature openssh
```

- Use the `win_chocolatey` Ansible module:

```
- name: install the Win32-OpenSSH service
  win_chocolatey:
    name: openssh
    package_params: /SSHServerFeature
    state: present
```

- Install an Ansible Galaxy role for example jborean93.win_openssh:

```
ansible-galaxy install jborean93.win_openssh
```

- Use the role in your playbook:

```
- name: install Win32-OpenSSH service
  hosts: windows
  gather_facts: false
  roles:
  - role: jborean93.win_openssh
    opt_openssh_setup_service: True
```

Note: `Win32-OpenSSH` is still a beta product and is constantly being updated to include new features and bugfixes. If you use SSH as a connection option for Windows, we highly recommend you install the latest version.

Configuring the Win32-OpenSSH shell

By default `Win32-OpenSSH` uses `cmd.exe` as a shell.

- To configure a different shell, use an Ansible playbook with a task to define the registry setting:

```
- name: set the default shell to PowerShell
  win_regedit:
    path: HKLM:\SOFTWARE\OpenSSH
    name: DefaultShell
    data: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
    type: string
    state: present
```

- To revert the settings back to the default shell:

```
- name: set the default shell to cmd
  win_regedit:
    path: HKLM:\SOFTWARE\OpenSSH
```

(continues on next page)

(continued from previous page)

```
    name: DefaultShell
    state: absent
```

Win32-OpenSSH Authentication

Win32-OpenSSH authentication with Windows is similar to SSH authentication on Unix/Linux hosts. You can use a plaintext password or SSH public key authentication.

For the key-based authentication:

- Add your public keys to an `authorized_key` file in the `.ssh` folder of the user's profile directory.
- Configure the SSH service using the `sshd_config` file.

When using SSH key authentication with Ansible, the remote session will not have access to user credentials and will fail when attempting to access a network resource. This is also known as the double-hop or credential delegation issue. To work around this problem:

- Use plaintext password authentication by setting the `ansible_password` variable.
- Use the `become` directive on the task with the credentials of the user that needs access to the remote resource.

Configuring Ansible for SSH on Windows

To configure Ansible to use SSH for Windows hosts, you must set two connection variables:

- set `ansible_connection` to `ssh`
- set `ansible_shell_type` to `cmd` or `powershell`

The `ansible_shell_type` variable should reflect the `DefaultShell` configured on the Windows host. Set `ansible_shell_type` to `cmd` for the default shell. Alternatively, set `ansible_shell_type` to `powershell` if you changed `DefaultShell` to PowerShell.

Known issues with SSH on Windows

Using SSH with Windows is experimental. Currently existing issues are:

- Win32-OpenSSH versions older than `v7.9.0.0p1-Beta` do not work when `powershell` is the shell type.
- While Secure Copy Protocol (SCP) should work, SSH File Transfer Protocol (SFTP) is the recommended mechanism to use when copying or fetching a file.

See also:

Ansible playbooks
    An introduction to playbooks

Ansible tips and tricks
    Tips and tricks for playbooks

List of Windows Modules
    Windows specific module list, all implemented in PowerShell

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

1.10.2 Using Ansible and Windows

When using Ansible to manage Windows, many of the syntax and rules that apply for Unix/Linux hosts also apply to Windows, but there are still some differences when it comes to components like path separators and OS-specific tasks. This document covers details specific to using Ansible for Windows.

Topics

Use Cases

Ansible can be used to orchestrate a multitude of tasks on Windows servers. Below are some examples and info about common tasks.

Installing Software

There are three main ways that Ansible can be used to install software:

- Using the `win_chocolatey` module. This sources the program data from the default public Chocolatey repository. Internal repositories can be used instead by setting the `source` option.
- Using the `win_package` module. This installs software using an MSI or .exe installer from a local/network path or URL.
- Using the `win_command` or `win_shell` module to run an installer manually.

The `win_chocolatey` module is recommended since it has the most complete logic for checking to see if a package has already been installed and is up-to-date.

Below are some examples of using all three options to install 7-Zip:

```yaml
# Install/uninstall with chocolatey
- name: Ensure 7-Zip is installed through Chocolatey
  win_chocolatey:
    name: 7zip
    state: present

- name: Ensure 7-Zip is not installed through Chocolatey
  win_chocolatey:
    name: 7zip
    state: absent

# Install/uninstall with win_package
- name: Download the 7-Zip package
  win_get_url:
    url: https://www.7-zip.org/a/7z1701-x64.msi
    dest: C:\temp\7z.msi

- name: Ensure 7-Zip is installed through win_package
  win_package:
    path: C:\temp\7z.msi
    state: present

- name: Ensure 7-Zip is not installed through win_package
  win_package:
    path: C:\temp\7z.msi
    state: absent

# Install/uninstall with win_command
- name: Download the 7-Zip package
  win_get_url:
    url: https://www.7-zip.org/a/7z1701-x64.msi
    dest: C:\temp\7z.msi

- name: Check if 7-Zip is already installed
  win_reg_stat:
    name: HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{23170F69-40C1-2702-
↪1701-000001000000}
  register: 7zip_installed

- name: Ensure 7-Zip is installed through win_command
  win_command: C:\Windows\System32\msiexec.exe /i C:\temp\7z.msi /qn /norestart
  when: 7zip_installed.exists == false

- name: Ensure 7-Zip is uninstalled through win_command
  win_command: C:\Windows\System32\msiexec.exe /x {23170F69-40C1-2702-1701-000001000000}␣
↪/qn /norestart
  when: 7zip_installed.exists == true
```

Some installers like Microsoft Office or SQL Server require credential delegation or access to components restricted by WinRM. The best method to bypass these issues is to use `become` with the task. With `become`, Ansible will run the installer as if it were run interactively on the host.

Note: Many installers do not properly pass back error information over WinRM. In these cases, if the install has been verified to work locally the recommended method is to use become.

Note: Some installers restart the WinRM or HTTP services, or cause them to become temporarily unavailable, making Ansible assume the system is unreachable.

Installing Updates

The `win_updates` and `win_hotfix` modules can be used to install updates or hotfixes on a host. The module `win_updates` is used to install multiple updates by category, while `win_hotfix` can be used to install a single update or hotfix file that has been downloaded locally.

Note: The `win_hotfix` module has a requirement that the DISM PowerShell cmdlets are present. These cmdlets were only added by default on Windows Server 2012 and newer and must be installed on older Windows hosts.

The following example shows how `win_updates` can be used:

```yaml
- name: Install all critical and security updates
  win_updates:
    category_names:
    - CriticalUpdates
    - SecurityUpdates
    state: installed
  register: update_result

- name: Reboot host if required
  win_reboot:
  when: update_result.reboot_required
```

The following example show how `win_hotfix` can be used to install a single update or hotfix:

```yaml
- name: Download KB3172729 for Server 2012 R2
  win_get_url:
    url: http://download.windowsupdate.com/d/msdownload/update/software/secu/2016/07/
↪windows8.1-kb3172729-x64_e8003822a7ef4705cbb65623b72fd3cec73fe222.msu
    dest: C:\temp\KB3172729.msu

- name: Install hotfix
  win_hotfix:
    hotfix_kb: KB3172729
    source: C:\temp\KB3172729.msu
    state: present
  register: hotfix_result

- name: Reboot host if required
  win_reboot:
  when: hotfix_result.reboot_required
```

Set Up Users and Groups

Ansible can be used to create Windows users and groups both locally and on a domain.

Local

The modules `win_user`, `win_group` and `win_group_membership` manage Windows users, groups and group memberships locally.

The following is an example of creating local accounts and groups that can access a folder on the same host:

```yaml
- name: Create local group to contain new users
  win_group:
    name: LocalGroup
    description: Allow access to C:\Development folder

- name: Create local user
  win_user:
    name: '{{ item.name }}'
    password: '{{ item.password }}'
    groups: LocalGroup
    update_password: false
    password_never_expires: true
  loop:
  - name: User1
    password: Password1
  - name: User2
    password: Password2

- name: Create Development folder
  win_file:
    path: C:\Development
    state: directory

- name: Set ACL of Development folder
  win_acl:
    path: C:\Development
    rights: FullControl
    state: present
    type: allow
    user: LocalGroup

- name: Remove parent inheritance of Development folder
  win_acl_inheritance:
    path: C:\Development
    reorganize: true
    state: absent
```

Domain

The modules win_domain_user and win_domain_group manages users and groups in a domain. The below is an example of ensuring a batch of domain users are created:

```
- name: Ensure each account is created
  win_domain_user:
    name: '{{ item.name }}'
    upn: '{{ item.name }}@MY.DOMAIN.COM'
    password: '{{ item.password }}'
    password_never_expires: false
    groups:
    - Test User
    - Application
    company: Ansible
    update_password: on_create
  loop:
  - name: Test User
    password: Password
  - name: Admin User
    password: SuperSecretPass01
  - name: Dev User
    password: '@fvr3IbFBujSRh!3hBg%wgFucD8^x8W5'
```

Running Commands

In cases where there is no appropriate module available for a task, a command or script can be run using the win_shell, win_command, raw, and script modules.

The raw module simply executes a Powershell command remotely. Since raw has none of the wrappers that Ansible typically uses, become, async and environment variables do not work.

The script module executes a script from the Ansible controller on one or more Windows hosts. Like raw, script currently does not support become, async, or environment variables.

The win_command module is used to execute a command which is either an executable or batch file, while the win_shell module is used to execute commands within a shell.

Choosing Command or Shell

The win_shell and win_command modules can both be used to execute a command or commands. The win_shell module is run within a shell-like process like PowerShell or cmd, so it has access to shell operators like <, >, |, ;, &&, and ||. Multi-lined commands can also be run in win_shell.

The win_command module simply runs a process outside of a shell. It can still run a shell command like mkdir or New-Item by passing the shell commands to a shell executable like cmd.exe or PowerShell.exe.

Here are some examples of using win_command and win_shell:

```
- name: Run a command under PowerShell
  win_shell: Get-Service -Name service | Stop-Service

- name: Run a command under cmd
```

(continues on next page)

(continued from previous page)

```
  win_shell: mkdir C:\temp
  args:
    executable: cmd.exe

- name: Run a multiple shell commands
  win_shell: |
    New-Item -Path C:\temp -ItemType Directory
    Remove-Item -Path C:\temp -Force -Recurse
    $path_info = Get-Item -Path C:\temp
    $path_info.FullName

- name: Run an executable using win_command
  win_command: whoami.exe

- name: Run a cmd command
  win_command: cmd.exe /c mkdir C:\temp

- name: Run a vbs script
  win_command: cscript.exe script.vbs
```

Note: Some commands like `mkdir`, `del`, and `copy` only exist in the CMD shell. To run them with `win_command` they must be prefixed with `cmd.exe /c`.

Argument Rules

When running a command through `win_command`, the standard Windows argument rules apply:

- Each argument is delimited by a white space, which can either be a space or a tab.
- An argument can be surrounded by double quotes `"`. Anything inside these quotes is interpreted as a single argument even if it contains whitespace.
- A double quote preceded by a backslash `\` is interpreted as just a double quote `"` and not as an argument delimiter.
- Backslashes are interpreted literally unless it immediately precedes double quotes; for example `\` == `\` and `\"` == `"`
- If an even number of backslashes is followed by a double quote, one backslash is used in the argument for every pair, and the double quote is used as a string delimiter for the argument.
- If an odd number of backslashes is followed by a double quote, one backslash is used in the argument for every pair, and the double quote is escaped and made a literal double quote in the argument.

With those rules in mind, here are some examples of quoting:

```
- win_command: C:\temp\executable.exe argument1 "argument 2" "C:\path\with space"
↪"double \"quoted\""

argv[0] = C:\temp\executable.exe
argv[1] = argument1
argv[2] = argument 2
argv[3] = C:\path\with space
argv[4] = double "quoted"
```

(continues on next page)

(continued from previous page)

```
- win_command: '"C:\Program Files\Program\program.exe" "escaped \\\" backslash" unquoted-
↪end-backslash\'

argv[0] = C:\Program Files\Program\program.exe
argv[1] = escaped \" backslash
argv[2] = unquoted-end-backslash\

# Due to YAML and Ansible parsing '\"' must be written as '{% raw %}\\{% endraw %}"'
- win_command: C:\temp\executable.exe C:\no\space\path "arg with end \ before end quote{
↪% raw %}\\{% endraw %}"

argv[0] = C:\temp\executable.exe
argv[1] = C:\no\space\path
argv[2] = arg with end \ before end quote\"
```

For more information, see escaping arguments.

Creating and Running a Scheduled Task

WinRM has some restrictions in place that cause errors when running certain commands. One way to bypass these restrictions is to run a command through a scheduled task. A scheduled task is a Windows component that provides the ability to run an executable on a schedule and under a different account.

Ansible version 2.5 added modules that make it easier to work with scheduled tasks in Windows. The following is an example of running a script as a scheduled task that deletes itself after running:

```
- name: Create scheduled task to run a process
  win_scheduled_task:
    name: adhoc-task
    username: SYSTEM
    actions:
    - path: PowerShell.exe
      arguments: |
        Start-Sleep -Seconds 30  # This isn't required, just here as a demonstration
        New-Item -Path C:\temp\test -ItemType Directory
    # Remove this action if the task shouldn't be deleted on completion
    - path: cmd.exe
      arguments: /c schtasks.exe /Delete /TN "adhoc-task" /F
    triggers:
    - type: registration

- name: Wait for the scheduled task to complete
  win_scheduled_task_stat:
    name: adhoc-task
  register: task_stat
  until: (task_stat.state is defined and task_stat.state.status != "TASK_STATE_RUNNING")␣
↪or (task_stat.task_exists == False)
  retries: 12
  delay: 10
```

Note: The modules used in the above example were updated/added in Ansible version 2.5.

Path Formatting for Windows

Windows differs from a traditional POSIX operating system in many ways. One of the major changes is the shift from `/` as the path separator to `\`. This can cause major issues with how playbooks are written, since `\` is often used as an escape character on POSIX systems.

Ansible allows two different styles of syntax; each deals with path separators for Windows differently:

YAML Style

When using the YAML syntax for tasks, the rules are well-defined by the YAML standard:

- When using a normal string (without quotes), YAML will not consider the backslash an escape character.
- When using single quotes `'`, YAML will not consider the backslash an escape character.
- When using double quotes `"`, the backslash is considered an escape character and needs to escaped with another backslash.

Note: You should only quote strings when it is absolutely necessary or required by YAML, and then use single quotes.

The YAML specification considers the following escape sequences:

- `\0`, `\\`, `\"`, `\_`, `\a`, `\b`, `\e`, `\f`, `\n`, `\r`, `\t`, `\v`, `\L`, `\N` and `\P` – Single character escape
- `<TAB>`, `<SPACE>`, `<NBSP>`, `<LNSP>`, `<PSP>` – Special characters
- `\x..` – 2-digit hex escape
- `\u....` – 4-digit hex escape
- `\U........` – 8-digit hex escape

Here are some examples on how to write Windows paths:

```
# GOOD
tempdir: C:\Windows\Temp

# WORKS
tempdir: 'C:\Windows\Temp'
tempdir: "C:\\Windows\\Temp"

# BAD, BUT SOMETIMES WORKS
tempdir: C:\\Windows\\Temp
tempdir: 'C:\\Windows\\Temp'
tempdir: C:/Windows/Temp
```

This is an example which will fail:

```
# FAILS
tempdir: "C:\Windows\Temp"
```

This example shows the use of single quotes when they are required:

```
---
- name: Copy tomcat config
  win_copy:
    src: log4j.xml
    dest: '{{tc_home}}\lib\log4j.xml'
```

Legacy key=value Style

The legacy `key=value` syntax is used on the command line for ad hoc commands, or inside playbooks. The use of this style is discouraged within playbooks because backslash characters need to be escaped, making playbooks harder to read. The legacy syntax depends on the specific implementation in Ansible, and quoting (both single and double) does not have any effect on how it is parsed by Ansible.

The Ansible key=value parser parse_kv() considers the following escape sequences:

- `\`, `'`, `"`, `\a`, `\b`, `\f`, `\n`, `\r`, `\t` and `\v` – Single character escape
- `\x..` – 2-digit hex escape
- `\u....` – 4-digit hex escape
- `\U........` – 8-digit hex escape
- `\N{...}` – Unicode character by name

This means that the backslash is an escape character for some sequences, and it is usually safer to escape a backslash when in this form.

Here are some examples of using Windows paths with the key=value style:

```
# GOOD
tempdir=C:\\Windows\\Temp

# WORKS
tempdir='C:\\Windows\\Temp'
tempdir="C:\\Windows\\Temp"

# BAD, BUT SOMETIMES WORKS
tempdir=C:\Windows\Temp
tempdir='C:\Windows\Temp'
tempdir="C:\Windows\Temp"
tempdir=C:/Windows/Temp

# FAILS
tempdir=C:\Windows\temp
tempdir='C:\Windows\temp'
tempdir="C:\Windows\temp"
```

The failing examples don't fail outright but will substitute `\t` with the `<TAB>` character resulting in `tempdir` being `C:\Windows<TAB>emp`.

Limitations

Some things you cannot do with Ansible and Windows are:

- Upgrade PowerShell
- Interact with the WinRM listeners

Because WinRM is reliant on the services being online and running during normal operations, you cannot upgrade PowerShell or interact with WinRM listeners with Ansible. Both of these actions will cause the connection to fail. This can technically be avoided by using `async` or a scheduled task, but those methods are fragile if the process it runs breaks the underlying connection Ansible uses, and are best left to the bootstrapping process or before an image is created.

Developing Windows Modules

Because Ansible modules for Windows are written in PowerShell, the development guides for Windows modules differ substantially from those for standard standard modules. Please see *Windows module development walkthrough* for more information.

See also:

Ansible playbooks
An introduction to playbooks

Ansible tips and tricks
Tips and tricks for playbooks

List of Windows Modules
Windows specific module list, all implemented in PowerShell

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

1.10.3 Windows Remote Management

Unlike Linux/Unix hosts, which use SSH by default, Windows hosts are configured with WinRM. This topic covers how to configure and use WinRM with Ansible.

What is WinRM?

WinRM is a management protocol used by Windows to remotely communicate with another server. It is a SOAP-based protocol that communicates over HTTP/HTTPS, and is included in all recent Windows operating systems. Since Windows Server 2012, WinRM has been enabled by default, but in most cases extra configuration is required to use WinRM with Ansible.

Ansible uses the pywinrm package to communicate with Windows servers over WinRM. It is not installed by default with the Ansible package, but can be installed by running the following:

```
pip install "pywinrm>=0.3.0"
```

Note: on distributions with multiple python versions, use pip2 or pip2.x, where x matches the python minor version Ansible is running under.

Warning: Using the `winrm` or `psrp` connection plugins in Ansible on MacOS in the latest releases typically fail. This is a known problem that occurs deep within the Python stack and cannot be changed by Ansible. The only workaround today is to set the environment variable `no_proxy=*` and avoid using Kerberos auth.

WinRM authentication options

When connecting to a Windows host, there are several different options that can be used when authenticating with an account. The authentication type may be set on inventory hosts or groups with the `ansible_winrm_transport` variable.

The following matrix is a high level overview of the options:

Option	Local Accounts	Active Directory Accounts	Credential Delegation	HTTP Encryption
Basic	Yes	No	No	No
Certificate	Yes	No	No	No
Kerberos	No	Yes	Yes	Yes
NTLM	Yes	Yes	No	Yes
CredSSP	Yes	Yes	Yes	Yes

Basic

Basic authentication is one of the simplest authentication options to use, but is also the most insecure. This is because the username and password are simply base64 encoded, and if a secure channel is not in use (eg, HTTPS) then it can be decoded by anyone. Basic authentication can only be used for local accounts (not domain accounts).

The following example shows host vars configured for basic authentication:

```
ansible_user: LocalUsername
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: basic
```

Basic authentication is not enabled by default on a Windows host but can be enabled by running the following in PowerShell:

```
Set-Item -Path WSMan:\localhost\Service\Auth\Basic -Value $true
```

Certificate

Certificate authentication uses certificates as keys similar to SSH key pairs, but the file format and key generation process is different.

The following example shows host vars configured for certificate authentication:

```
ansible_connection: winrm
ansible_winrm_cert_pem: /path/to/certificate/public/key.pem
ansible_winrm_cert_key_pem: /path/to/certificate/private/key.pem
ansible_winrm_transport: certificate
```

Certificate authentication is not enabled by default on a Windows host but can be enabled by running the following in PowerShell:

```
Set-Item -Path WSMan:\localhost\Service\Auth\Certificate -Value $true
```

Note: Encrypted private keys cannot be used as the urllib3 library that is used by Ansible for WinRM does not support this functionality.

Note: Certificate authentication does not work with a TLS 1.3 connection.

.._winrm_certificate_generate:

Generate a Certificate

A certificate must be generated before it can be mapped to a local user. This can be done using one of the following methods:

- OpenSSL
- PowerShell, using the `New-SelfSignedCertificate` cmdlet
- Active Directory Certificate Services

Active Directory Certificate Services is beyond of scope in this documentation but may be the best option to use when running in a domain environment. For more information, see the Active Directory Certificate Services documentation.

Note: Using the PowerShell cmdlet `New-SelfSignedCertificate` to generate a certificate for authentication only works when being generated from a Windows 10 or Windows Server 2012 R2 host or later. OpenSSL is still required to extract the private key from the PFX certificate to a PEM file for Ansible to use.

To generate a certificate with `OpenSSL`:

```
# Set the name of the local user that will have the key mapped to
USERNAME="username"

cat > openssl.conf << EOL
distinguished_name = req_distinguished_name
[req_distinguished_name]
[v3_req_client]
extendedKeyUsage = clientAuth
subjectAltName = otherName:1.3.6.1.4.1.311.20.2.3;UTF8:$USERNAME@localhost
EOL

export OPENSSL_CONF=openssl.conf
openssl req -x509 -nodes -days 3650 -newkey rsa:2048 -out cert.pem -outform PEM -keyout␣
↪cert_key.pem -subj "/CN=$USERNAME" -extensions v3_req_client
rm openssl.conf
```

To generate a certificate with `New-SelfSignedCertificate`:

```
# Set the name of the local user that will have the key mapped
$username = "username"
$output_path = "C:\temp"

# Instead of generating a file, the cert will be added to the personal
# LocalComputer folder in the certificate store
$cert = New-SelfSignedCertificate -Type Custom `
    -Subject "CN=$username" `
    -TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.2","2.5.29.17={text}upn=
↪$username@localhost") `
    -KeyUsage DigitalSignature,KeyEncipherment `
    -KeyAlgorithm RSA `
    -KeyLength 2048

# Export the public key
$pem_output = @()
```

(continues on next page)

(continued from previous page)

```
$pem_output += "-----BEGIN CERTIFICATE-----"
$pem_output += [System.Convert]::ToBase64String($cert.RawData) -replace ".{64}", "$&`n"
$pem_output += "-----END CERTIFICATE-----"
[System.IO.File]::WriteAllLines("$output_path\cert.pem", $pem_output)

# Export the private key in a PFX file
[System.IO.File]::WriteAllBytes("$output_path\cert.pfx", $cert.Export("Pfx"))
```

Note: To convert the PFX file to a private key that pywinrm can use, run the following command with OpenSSL `openssl pkcs12 -in cert.pfx -nocerts -nodes -out cert_key.pem -passin pass: -passout pass:`

Import a Certificate to the Certificate Store

Once a certificate has been generated, the issuing certificate needs to be imported into the `Trusted Root Certificate Authorities` of the `LocalMachine` store, and the client certificate public key must be present in the `Trusted People` folder of the `LocalMachine` store. For this example, both the issuing certificate and public key are the same.

Following example shows how to import the issuing certificate:

```
$cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.
↪X509Certificate2 "cert.pem"

$store_name = [System.Security.Cryptography.X509Certificates.StoreName]::Root
$store_location = [System.Security.Cryptography.X509Certificates.
↪StoreLocation]::LocalMachine
$store = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Store -
↪ArgumentList $store_name, $store_location
$store.Open("MaxAllowed")
$store.Add($cert)
$store.Close()
```

Note: If using ADCS to generate the certificate, then the issuing certificate will already be imported and this step can be skipped.

The code to import the client certificate public key is:

```
$cert = New-Object -TypeName System.Security.Cryptography.X509Certificates.
↪X509Certificate2 "cert.pem"

$store_name = [System.Security.Cryptography.X509Certificates.StoreName]::TrustedPeople
$store_location = [System.Security.Cryptography.X509Certificates.
↪StoreLocation]::LocalMachine
$store = New-Object -TypeName System.Security.Cryptography.X509Certificates.X509Store -
↪ArgumentList $store_name, $store_location
$store.Open("MaxAllowed")
$store.Add($cert)
$store.Close()
```

Mapping a Certificate to an Account

Once the certificate has been imported, map it to the local user account:

```
$username = "username"
$password = ConvertTo-SecureString -String "password" -AsPlainText -Force
$credential = New-Object -TypeName System.Management.Automation.PSCredential -
↪ArgumentList $username, $password

# This is the issuer thumbprint which in the case of a self generated cert
# is the public key thumbprint, additional logic may be required for other
# scenarios
$thumbprint = (Get-ChildItem -Path cert:\LocalMachine\root | Where-Object { $_.Subject -
↪eq "CN=$username" }).Thumbprint

New-Item -Path WSMan:\localhost\ClientCertificate `
    -Subject "$username@localhost" `
    -URI * `
    -Issuer $thumbprint `
    -Credential $credential `
    -Force
```

Once this is complete, the hostvar `ansible_winrm_cert_pem` should be set to the path of the public key and the `ansible_winrm_cert_key_pem` variable should be set to the path of the private key.

NTLM

NTLM is an older authentication mechanism used by Microsoft that can support both local and domain accounts. NTLM is enabled by default on the WinRM service, so no setup is required before using it.

NTLM is the easiest authentication protocol to use and is more secure than `Basic` authentication. If running in a domain environment, `Kerberos` should be used instead of NTLM.

Kerberos has several advantages over using NTLM:

- NTLM is an older protocol and does not support newer encryption protocols.
- NTLM is slower to authenticate because it requires more round trips to the host in the authentication stage.
- Unlike Kerberos, NTLM does not allow credential delegation.

This example shows host variables configured to use NTLM authentication:

```
ansible_user: LocalUsername
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: ntlm
```

Kerberos

Kerberos is the recommended authentication option to use when running in a domain environment. Kerberos supports features like credential delegation and message encryption over HTTP and is one of the more secure options that is available through WinRM.

Kerberos requires some additional setup work on the Ansible host before it can be used properly.

The following example shows host vars configured for Kerberos authentication:

```
ansible_user: username@MY.DOMAIN.COM
ansible_password: Password
ansible_connection: winrm
ansible_port: 5985
ansible_winrm_transport: kerberos
```

As of Ansible version 2.3, the Kerberos ticket will be created based on `ansible_user` and `ansible_password`. If running on an older version of Ansible or when `ansible_winrm_kinit_mode` is `manual`, a Kerberos ticket must already be obtained. See below for more details.

There are some extra host variables that can be set:

```
ansible_winrm_kinit_mode: managed/manual (manual means Ansible will not obtain a ticket)
ansible_winrm_kinit_cmd: the kinit binary to use to obtain a Kerberos ticket (default to␣
↪kinit)
ansible_winrm_service: overrides the SPN prefix that is used, the default is ``HTTP``␣
↪and should rarely ever need changing
ansible_winrm_kerberos_delegation: allows the credentials to traverse multiple hops
ansible_winrm_kerberos_hostname_override: the hostname to be used for the kerberos␣
↪exchange
```

Installing the Kerberos Library

Some system dependencies that must be installed prior to using Kerberos. The script below lists the dependencies based on the distro:

```
# Through Yum (RHEL/Centos/Fedora for the older version)
yum -y install gcc python-devel krb5-devel krb5-libs krb5-workstation

# Through DNF (RHEL/Centos/Fedora for the newer version)
dnf -y install gcc python3-devel krb5-devel krb5-libs krb5-workstation

# Through Apt (Ubuntu)
sudo apt-get install python-dev libkrb5-dev krb5-user

# Through Portage (Gentoo)
emerge -av app-crypt/mit-krb5
emerge -av dev-python/setuptools

# Through Pkg (FreeBSD)
sudo pkg install security/krb5

# Through OpenCSW (Solaris)
```

(continues on next page)

(continued from previous page)

```
pkgadd -d http://get.opencsw.org/now
/opt/csw/bin/pkgutil -U
/opt/csw/bin/pkgutil -y -i libkrb5_3

# Through Pacman (Arch Linux)
pacman -S krb5
```

Once the dependencies have been installed, the `python-kerberos` wrapper can be install using `pip`:

```
pip install pywinrm[kerberos]
```

Note: While Ansible has supported Kerberos auth through `pywinrm` for some time, optional features or more secure options may only be available in newer versions of the `pywinrm` and/or `pykerberos` libraries. It is recommended you upgrade each version to the latest available to resolve any warnings or errors. This can be done through tools like `pip` or a system package manager like `dnf`, `yum`, `apt` but the package names and versions available may differ between tools.

Configuring Host Kerberos

Once the dependencies have been installed, Kerberos needs to be configured so that it can communicate with a domain. This configuration is done through the `/etc/krb5.conf` file, which is installed with the packages in the script above.

To configure Kerberos, in the section that starts with:

```
[realms]
```

Add the full domain name and the fully qualified domain names of the primary and secondary Active Directory domain controllers. It should look something like this:

```
[realms]
    MY.DOMAIN.COM = {
        kdc = domain-controller1.my.domain.com
        kdc = domain-controller2.my.domain.com
    }
```

In the section that starts with:

```
[domain_realm]
```

Add a line like the following for each domain that Ansible needs access for:

```
[domain_realm]
    .my.domain.com = MY.DOMAIN.COM
```

You can configure other settings in this file such as the default domain. See krb5.conf for more details.

Automatic Kerberos Ticket Management

Ansible version 2.3 and later defaults to automatically managing Kerberos tickets when both `ansible_user` and `ansible_password` are specified for a host. In this process, a new ticket is created in a temporary credential cache for each host. This is done before each task executes to minimize the chance of ticket expiration. The temporary credential caches are deleted after each task completes and will not interfere with the default credential cache.

To disable automatic ticket management, set `ansible_winrm_kinit_mode=manual` through the inventory.

Automatic ticket management requires a standard `kinit` binary on the control host system path. To specify a different location or binary name, set the `ansible_winrm_kinit_cmd` hostvar to the fully qualified path to a MIT krbv5 `kinit`-compatible binary.

Manual Kerberos Ticket Management

To manually manage Kerberos tickets, the `kinit` binary is used. To obtain a new ticket the following command is used:

```
kinit username@MY.DOMAIN.COM
```

Note: The domain must match the configured Kerberos realm exactly, and must be in upper case.

To see what tickets (if any) have been acquired, use the following command:

```
klist
```

To destroy all the tickets that have been acquired, use the following command:

```
kdestroy
```

Troubleshooting Kerberos

Kerberos is reliant on a properly-configured environment to work. To troubleshoot Kerberos issues, ensure that:

- The hostname set for the Windows host is the FQDN and not an IP address. * If you connect using an IP address you will get the error message *Server not found in Kerberos database*. * To determine if you are connecting using an IP address or an FQDN run your playbook (or call the `win_ping` module) using the *-vvv* flag.
- The forward and reverse DNS lookups are working properly in the domain. To test this, ping the windows host by name and then use the ip address returned with `nslookup`. The same name should be returned when using `nslookup` on the IP address.
- The Ansible host's clock is synchronized with the domain controller. Kerberos is time sensitive, and a little clock drift can cause the ticket generation process to fail.
- Ensure that the fully qualified domain name for the domain is configured in the `krb5.conf` file. To check this, run:

  ```
  kinit -C username@MY.DOMAIN.COM
  klist
  ```

  If the domain name returned by `klist` is different from the one requested, an alias is being used. The `krb5.conf` file needs to be updated so that the fully qualified domain name is used and not an alias.

- If the default kerberos tooling has been replaced or modified (some IdM solutions may do this), this may cause issues when installing or upgrading the Python Kerberos library. As of the time of this writing, this library is called `pykerberos` and is known to work with both MIT and Heimdal Kerberos libraries. To resolve `pykerberos` installation issues, ensure the system dependencies for Kerberos have been met (see: *Installing the Kerberos Library*), remove any custom Kerberos tooling paths from the PATH environment variable, and retry the installation of Python Kerberos library package.

CredSSP

CredSSP authentication is a newer authentication protocol that allows credential delegation. This is achieved by encrypting the username and password after authentication has succeeded and sending that to the server using the CredSSP protocol.

Because the username and password are sent to the server to be used for double hop authentication, ensure that the hosts that the Windows host communicates with are not compromised and are trusted.

CredSSP can be used for both local and domain accounts and also supports message encryption over HTTP.

To use CredSSP authentication, the host vars are configured like so:

```
ansible_user: Username
ansible_password: Password
ansible_connection: winrm
ansible_winrm_transport: credssp
```

There are some extra host variables that can be set as shown below:

```
ansible_winrm_credssp_disable_tlsv1_2: when true, will not use TLS 1.2 in the CredSSP␣
↪auth process
```

CredSSP authentication is not enabled by default on a Windows host, but can be enabled by running the following in PowerShell:

```
Enable-WSManCredSSP -Role Server -Force
```

Installing CredSSP Library

The `requests-credssp` wrapper can be installed using `pip`:

```
pip install pywinrm[credssp]
```

CredSSP and TLS 1.2

By default the `requests-credssp` library is configured to authenticate over the TLS 1.2 protocol. TLS 1.2 is installed and enabled by default for Windows Server 2012 and Windows 8 and more recent releases.

There are two ways that older hosts can be used with CredSSP:

- Install and enable a hotfix to enable TLS 1.2 support (recommended for Server 2008 R2 and Windows 7).
- Set `ansible_winrm_credssp_disable_tlsv1_2=True` in the inventory to run over TLS 1.0. This is the only option when connecting to Windows Server 2008, which has no way of supporting TLS 1.2

See *TLS 1.2 Support* for more information on how to enable TLS 1.2 on the Windows host.

Set CredSSP Certificate

CredSSP works by encrypting the credentials through the TLS protocol and uses a self-signed certificate by default. The `CertificateThumbprint` option under the WinRM service configuration can be used to specify the thumbprint of another certificate.

Note: This certificate configuration is independent of the WinRM listener certificate. With CredSSP, message transport still occurs over the WinRM listener, but the TLS-encrypted messages inside the channel use the service-level certificate.

To explicitly set the certificate to use for CredSSP:

```
# Note the value $certificate_thumbprint will be different in each
# situation, this needs to be set based on the cert that is used.
$certificate_thumbprint = "7C8DCBD5427AFEE6560F4AF524E325915F51172C"

# Set the thumbprint value
Set-Item -Path WSMan:\localhost\Service\CertificateThumbprint -Value $certificate_
↪thumbprint
```

Non-Administrator Accounts

WinRM is configured by default to only allow connections from accounts in the local `Administrators` group. This can be changed by running:

```
winrm configSDDL default
```

This will display an ACL editor, where new users or groups may be added. To run commands over WinRM, users and groups must have at least the `Read` and `Execute` permissions enabled.

While non-administrative accounts can be used with WinRM, most typical server administration tasks require some level of administrative access, so the utility is usually limited.

WinRM Encryption

By default WinRM will fail to work when running over an unencrypted channel. The WinRM protocol considers the channel to be encrypted if using TLS over HTTP (HTTPS) or using message level encryption. Using WinRM with TLS is the recommended option as it works with all authentication options, but requires a certificate to be created and used on the WinRM listener.

If in a domain environment, ADCS can create a certificate for the host that is issued by the domain itself.

If using HTTPS is not an option, then HTTP can be used when the authentication option is `NTLM`, `Kerberos` or `CredSSP`. These protocols will encrypt the WinRM payload with their own encryption method before sending it to the server. The message-level encryption is not used when running over HTTPS because the encryption uses the more secure TLS protocol instead. If both transport and message encryption is required, set `ansible_winrm_message_encryption=always` in the host vars.

Note: Message encryption over HTTP requires pywinrm>=0.3.0.

A last resort is to disable the encryption requirement on the Windows host. This should only be used for development and debugging purposes, as anything sent from Ansible can be viewed, manipulated and also the remote session can completely be taken over by anyone on the same network. To disable the encryption requirement:

```
Set-Item -Path WSMan:\localhost\Service\AllowUnencrypted -Value $true
```

Note: Do not disable the encryption check unless it is absolutely required. Doing so could allow sensitive information like credentials and files to be intercepted by others on the network.

Inventory Options

Ansible's Windows support relies on a few standard variables to indicate the username, password, and connection type of the remote hosts. These variables are most easily set up in the inventory, but can be set on the `host_vars/group_vars` level.

When setting up the inventory, the following variables are required:

```
# It is suggested that these be encrypted with ansible-vault:
# ansible-vault edit group_vars/windows.yml
ansible_connection: winrm

# May also be passed on the command-line through --user
ansible_user: Administrator

# May also be supplied at runtime with --ask-pass
ansible_password: SecretPasswordGoesHere
```

Using the variables above, Ansible will connect to the Windows host with Basic authentication through HTTPS. If `ansible_user` has a UPN value like `username@MY.DOMAIN.COM` then the authentication option will automatically attempt to use Kerberos unless `ansible_winrm_transport` has been set to something other than `kerberos`.

The following custom inventory variables are also supported for additional configuration of WinRM connections:

- `ansible_port`: The port WinRM will run over, HTTPS is `5986` which is the default while HTTP is `5985`
- `ansible_winrm_scheme`: Specify the connection scheme (`http` or `https`) to use for the WinRM connection. Ansible uses `https` by default unless `ansible_port` is `5985`
- `ansible_winrm_path`: Specify an alternate path to the WinRM endpoint, Ansible uses `/wsman` by default
- `ansible_winrm_realm`: Specify the realm to use for Kerberos authentication. If `ansible_user` contains `@`, Ansible will use the part of the username after `@` by default
- `ansible_winrm_transport`: Specify one or more authentication transport options as a comma-separated list. By default, Ansible will use `kerberos, basic` if the `kerberos` module is installed and a realm is defined, otherwise it will be `plaintext`
- `ansible_winrm_server_cert_validation`: Specify the server certificate validation mode (`ignore` or `validate`). Ansible defaults to `validate` on Python 2.7.9 and higher, which will result in certificate validation errors against the Windows self-signed certificates. Unless verifiable certificates have been configured on the WinRM listeners, this should be set to `ignore`
- `ansible_winrm_operation_timeout_sec`: Increase the default timeout for WinRM operations, Ansible uses `20` by default
- `ansible_winrm_read_timeout_sec`: Increase the WinRM read timeout, Ansible uses `30` by default. Useful if there are intermittent network issues and read timeout errors keep occurring

- `ansible_winrm_message_encryption`: Specify the message encryption operation (`auto`, `always`, `never`) to use, Ansible uses `auto` by default. `auto` means message encryption is only used when `ansible_winrm_scheme` is `http` and `ansible_winrm_transport` supports message encryption. `always` means message encryption will always be used and `never` means message encryption will never be used
- `ansible_winrm_ca_trust_path`: Used to specify a different cacert container than the one used in the `certifi` module. See the HTTPS Certificate Validation section for more details.
- `ansible_winrm_send_cbt`: When using `ntlm` or `kerberos` over HTTPS, the authentication library will try to send channel binding tokens to mitigate against man in the middle attacks. This flag controls whether these bindings will be sent or not (default: `yes`).
- `ansible_winrm_*`: Any additional keyword arguments supported by `winrm.Protocol` may be provided in place of `*`

In addition, there are also specific variables that need to be set for each authentication option. See the section on authentication above for more information.

Note: Ansible 2.0 has deprecated the "ssh" from `ansible_ssh_user`, `ansible_ssh_pass`, `ansible_ssh_host`, and `ansible_ssh_port` to become `ansible_user`, `ansible_password`, `ansible_host`, and `ansible_port`. If using a version of Ansible prior to 2.0, the older style (`ansible_ssh_*`) should be used instead. The shorter variables are ignored, without warning, in older versions of Ansible.

Note: `ansible_winrm_message_encryption` is different from transport encryption done over TLS. The WinRM payload is still encrypted with TLS when run over HTTPS, even if `ansible_winrm_message_encryption=never`.

IPv6 Addresses

IPv6 addresses can be used instead of IPv4 addresses or hostnames. This option is normally set in an inventory. Ansible will attempt to parse the address using the ipaddress package and pass to pywinrm correctly.

When defining a host using an IPv6 address, just add the IPv6 address as you would an IPv4 address or hostname:

```ini
[windows-server]
2001:db8::1

[windows-server:vars]
ansible_user=username
ansible_password=password
ansible_connection=winrm
```

Note: The ipaddress library is only included by default in Python 3.x. To use IPv6 addresses in Python 2.7, make sure to run `pip install ipaddress` which installs a backported package.

HTTPS Certificate Validation

As part of the TLS protocol, the certificate is validated to ensure the host matches the subject and the client trusts the issuer of the server certificate. When using a self-signed certificate or setting `ansible_winrm_server_cert_validation: ignore` these security mechanisms are bypassed. While self signed certificates will always need the `ignore` flag, certificates that have been issued from a certificate authority can still be validated.

One of the more common ways of setting up a HTTPS listener in a domain environment is to use Active Directory Certificate Service (AD CS). AD CS is used to generate signed certificates from a Certificate Signing Request (CSR). If the WinRM HTTPS listener is using a certificate that has been signed by another authority, like AD CS, then Ansible can be set up to trust that issuer as part of the TLS handshake.

To get Ansible to trust a Certificate Authority (CA) like AD CS, the issuer certificate of the CA can be exported as a PEM encoded certificate. This certificate can then be copied locally to the Ansible controller and used as a source of certificate validation, otherwise known as a CA chain.

The CA chain can contain a single or multiple issuer certificates and each entry is contained on a new line. To then use the custom CA chain as part of the validation process, set `ansible_winrm_ca_trust_path` to the path of the file. If this variable is not set, the default CA chain is used instead which is located in the install path of the Python package certifi.

Note: Each HTTP call is done by the Python requests library which does not use the systems built-in certificate store as a trust authority. Certificate validation will fail if the server's certificate issuer is only added to the system's truststore.

TLS 1.2 Support

As WinRM runs over the HTTP protocol, using HTTPS means that the TLS protocol is used to encrypt the WinRM messages. TLS will automatically attempt to negotiate the best protocol and cipher suite that is available to both the client and the server. If a match cannot be found then Ansible will error out with a message similar to:

```
HTTPSConnectionPool(host='server', port=5986): Max retries exceeded with url: /wsman␣
↪(Caused by SSLError(SSLError(1, '[SSL: UNSUPPORTED_PROTOCOL] unsupported protocol (_
↪ssl.c:1056)')))
```

Commonly this is when the Windows host has not been configured to support TLS v1.2 but it could also mean the Ansible controller has an older OpenSSL version installed.

Windows 8 and Windows Server 2012 come with TLS v1.2 installed and enabled by default but older hosts, like Server 2008 R2 and Windows 7, have to be enabled manually.

Note: There is a bug with the TLS 1.2 patch for Server 2008 which will stop Ansible from connecting to the Windows host. This means that Server 2008 cannot be configured to use TLS 1.2. Server 2008 R2 and Windows 7 are not affected by this issue and can use TLS 1.2.

To verify what protocol the Windows host supports, you can run the following command on the Ansible controller:

```
openssl s_client -connect <hostname>:5986
```

The output will contain information about the TLS session and the `Protocol` line will display the version that was negotiated:

```
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1
    Cipher    : ECDHE-RSA-AES256-SHA
    Session-ID: 962A00001C95D2A601BE1CCFA7831B85A7EEE897AECDBF3D9ECD4A3BE4F6AC9B
    Session-ID-ctx:
    Master-Key: ....
    Start Time: 1552976474
    Timeout   : 7200 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---

New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: AE16000050DA9FD44D03BB8839B64449805D9E43DBD670346D3D9E05D1AEEA84
    Session-ID-ctx:
    Master-Key: ....
    Start Time: 1552976538
    Timeout   : 7200 (sec)
    Verify return code: 21 (unable to verify the first certificate)
```

If the host is returning TLSv1 then it should be configured so that TLS v1.2 is enable. You can do this by running the following PowerShell script:

```
Function Enable-TLS12 {
    param(
        [ValidateSet("Server", "Client")]
        [String]$Component = "Server"
    )

    $protocols_path = 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\
↪Protocols'
    New-Item -Path "$protocols_path\TLS 1.2\$Component" -Force
    New-ItemProperty -Path "$protocols_path\TLS 1.2\$Component" -Name Enabled -Value 1 -
↪Type DWORD -Force
    New-ItemProperty -Path "$protocols_path\TLS 1.2\$Component" -Name DisabledByDefault -
↪Value 0 -Type DWORD -Force
}

Enable-TLS12 -Component Server
```

(continues on next page)

(continued from previous page)

```
# Not required but highly recommended to enable the Client side TLS 1.2 components
Enable-TLS12 -Component Client

Restart-Computer
```

The below Ansible tasks can also be used to enable TLS v1.2:

```
- name: enable TLSv1.2 support
  win_regedit:
    path: HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\
↪TLS 1.2\{{ item.type }}
    name: '{{ item.property }}'
    data: '{{ item.value }}'
    type: dword
    state: present
  register: enable_tls12
  loop:
  - type: Server
    property: Enabled
    value: 1
  - type: Server
    property: DisabledByDefault
    value: 0
  - type: Client
    property: Enabled
    value: 1
  - type: Client
    property: DisabledByDefault
    value: 0

- name: reboot if TLS config was applied
  win_reboot:
  when: enable_tls12 is changed
```

There are other ways to configure the TLS protocols as well as the cipher suites that are offered by the Windows host. One tool that can give you a GUI to manage these settings is IIS Crypto from Nartac Software.

WinRM limitations

Due to the design of the WinRM protocol , there are a few limitations when using WinRM that can cause issues when creating playbooks for Ansible. These include:

- Credentials are not delegated for most authentication types, which causes authentication errors when accessing network resources or installing certain programs.
- Many calls to the Windows Update API are blocked when running over WinRM.
- Some programs fail to install with WinRM due to no credential delegation or because they access forbidden Windows API like WUA over WinRM.
- Commands under WinRM are done under a non-interactive session, which can prevent certain commands or executables from running.
- You cannot run a process that interacts with `DPAPI`, which is used by some installers (like Microsoft SQL Server).

Some of these limitations can be mitigated by doing one of the following:

- Set `ansible_winrm_transport` to `credssp` or `kerberos` (with `ansible_winrm_kerberos_delegation=true`) to bypass the double hop issue and access network resources
- Use `become` to bypass all WinRM restrictions and run a command as it would locally. Unlike using an authentication transport like `credssp`, this will also remove the non-interactive restriction and API restrictions like WUA and DPAPI
- Use a scheduled task to run a command which can be created with the `win_scheduled_task` module. Like `become`, this bypasses all WinRM restrictions but can only run a command and not modules.

See also:

Ansible playbooks
An introduction to playbooks

Ansible tips and tricks
Tips and tricks for playbooks

List of Windows Modules
Windows specific module list, all implemented in PowerShell

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

1.10.4 Desired State Configuration

Topics

What is Desired State Configuration?

Desired State Configuration, or DSC, is a tool built into PowerShell that can be used to define a Windows host setup through code. The overall purpose of DSC is the same as Ansible, it is just executed in a different manner. Since Ansible 2.4, the `win_dsc` module has been added and can be used to take advantage of existing DSC resources when interacting with a Windows host.

More details on DSC can be viewed at DSC Overview.

Host Requirements

To use the `win_dsc` module, a Windows host must have PowerShell v5.0 or newer installed. All supported hosts can be upgraded to PowerShell v5.

Once the PowerShell requirements have been met, using DSC is as simple as creating a task with the `win_dsc` module.

Why Use DSC?

DSC and Ansible modules have a common goal which is to define and ensure the state of a resource. Because of this, resources like the DSC File resource and Ansible `win_file` can be used to achieve the same result. Deciding which to use depends on the scenario.

Reasons for using an Ansible module over a DSC resource:

- The host does not support PowerShell v5.0, or it cannot easily be upgraded
- The DSC resource does not offer a feature present in an Ansible module. For example, win_regedit can manage the `REG_NONE` property type, while the DSC `Registry` resource cannot
- DSC resources have limited check mode support, while some Ansible modules have better checks
- DSC resources do not support diff mode, while some Ansible modules do
- Custom resources require further installation steps to be run on the host beforehand, while Ansible modules are built-in to Ansible
- There are bugs in a DSC resource where an Ansible module works

Reasons for using a DSC resource over an Ansible module:

- The Ansible module does not support a feature present in a DSC resource
- There is no Ansible module available
- There are bugs in an existing Ansible module

In the end, it doesn't matter whether the task is performed with DSC or an Ansible module; what matters is that the task is performed correctly and the playbooks are still readable. If you have more experience with DSC over Ansible and it does the job, just use DSC for that task.

How to Use DSC?

The `win_dsc` module takes in a free-form of options so that it changes according to the resource it is managing. A list of built-in resources can be found at resources.

Using the Registry resource as an example, this is the DSC definition as documented by Microsoft:

```
Registry [string] #ResourceName
{
    Key = [string]
    ValueName = [string]
    [ Ensure = [string] { Enable | Disable }  ]
    [ Force =  [bool]   ]
    [ Hex = [bool] ]
    [ DependsOn = [string[]] ]
    [ ValueData = [string[]] ]
    [ ValueType = [string] { Binary | Dword | ExpandString | MultiString | Qword |␣
↪String }  ]
}
```

When defining the task, `resource_name` must be set to the DSC resource being used - in this case, the `resource_name` should be set to `Registry`. The `module_version` can refer to a specific version of the DSC resource installed; if left blank it will default to the latest version. The other options are parameters that are used to define the resource, such as `Key` and `ValueName`. While the options in the task are not case sensitive, keeping the case as-is is recommended because it makes it easier to distinguish DSC resource options from Ansible's `win_dsc` options.

This is what the Ansible task version of the above DSC Registry resource would look like:

```
- name: Use win_dsc module with the Registry DSC resource
  win_dsc:
    resource_name: Registry
    Ensure: Present
    Key: HKEY_LOCAL_MACHINE\SOFTWARE\ExampleKey
    ValueName: TestValue
    ValueData: TestData
```

Starting in Ansible 2.8, the `win_dsc` module automatically validates the input options from Ansible with the DSC definition. This means Ansible will fail if the option name is incorrect, a mandatory option is not set, or the value is not a valid choice. When running Ansible with a verbosity level of 3 or more (`-vvv`), the return value will contain the possible invocation options based on the `resource_name` specified. Here is an example of the invocation output for the above `Registry` task:

```
changed: [2016] => {
    "changed": true,
    "invocation": {
        "module_args": {
            "DependsOn": null,
            "Ensure": "Present",
            "Force": null,
            "Hex": null,
            "Key": "HKEY_LOCAL_MACHINE\\SOFTWARE\\ExampleKey",
            "PsDscRunAsCredential_password": null,
            "PsDscRunAsCredential_username": null,
            "ValueData": [
                "TestData"
```

(continues on next page)

(continued from previous page)

```
            ],
            "ValueName": "TestValue",
            "ValueType": null,
            "module_version": "latest",
            "resource_name": "Registry"
        }
    },
    "module_version": "1.1",
    "reboot_required": false,
    "verbose_set": [
        "Perform operation 'Invoke CimMethod' with following parameters, ''methodName' =␣
↪ResourceSet,'className' = MSFT_DSCLocalConfigurationManager,'namespaceName' = root/
↪Microsoft/Windows/DesiredStateConfiguration'.",
        "An LCM method call arrived from computer SERVER2016 with user sid S-1-5-21-
↪3088887838-4058132883-1884671576-1105.",
        "[SERVER2016]: LCM:  [ Start  Set      ]  [[Registry]DirectResourceAccess]",
        "[SERVER2016]:                            [[Registry]DirectResourceAccess] (SET)␣
↪Create registry key 'HKLM:\\SOFTWARE\\ExampleKey'",
        "[SERVER2016]:                            [[Registry]DirectResourceAccess] (SET)␣
↪Set registry key value 'HKLM:\\SOFTWARE\\ExampleKey\\TestValue' to 'TestData' of type
↪'String'",
        "[SERVER2016]: LCM:  [ End    Set      ]  [[Registry]DirectResourceAccess]  in 0.
↪1930 seconds.",
        "[SERVER2016]: LCM:  [ End    Set      ]    in  0.2720 seconds.",
        "Operation 'Invoke CimMethod' complete.",
        "Time taken for configuration job to complete is 0.402 seconds"
    ],
    "verbose_test": [
        "Perform operation 'Invoke CimMethod' with following parameters, ''methodName' =␣
↪ResourceTest,'className' = MSFT_DSCLocalConfigurationManager,'namespaceName' = root/
↪Microsoft/Windows/DesiredStateConfiguration'.",
        "An LCM method call arrived from computer SERVER2016 with user sid S-1-5-21-
↪3088887838-4058132883-1884671576-1105.",
        "[SERVER2016]: LCM:  [ Start  Test     ]  [[Registry]DirectResourceAccess]",
        "[SERVER2016]:                            [[Registry]DirectResourceAccess]␣
↪Registry key 'HKLM:\\SOFTWARE\\ExampleKey' does not exist",
        "[SERVER2016]: LCM:  [ End    Test     ]  [[Registry]DirectResourceAccess] False␣
↪in 0.2510 seconds.",
        "[SERVER2016]: LCM:  [ End    Set      ]    in  0.3310 seconds.",
        "Operation 'Invoke CimMethod' complete.",
        "Time taken for configuration job to complete is 0.475 seconds"
    ]
}
```

The `invocation.module_args` key shows the actual values that were set as well as other possible values that were not set. Unfortunately, this will not show the default value for a DSC property, only what was set from the Ansible task. Any `*_password` option will be masked in the output for security reasons; if there are any other sensitive module options, set `no_log: True` on the task to stop all task output from being logged.

Property Types

Each DSC resource property has a type that is associated with it. Ansible will try to convert the defined options to the correct type during execution. For simple types like `[string]` and `[bool]`, this is a simple operation, but complex types like `[PSCredential]` or arrays (like `[string[]]`) require certain rules.

PSCredential

A `[PSCredential]` object is used to store credentials in a secure way, but Ansible has no way to serialize this over JSON. To set a DSC PSCredential property, the definition of that parameter should have two entries that are suffixed with `_username` and `_password` for the username and password, respectively. For example:

```yaml
PsDscRunAsCredential_username: '{{ ansible_user }}'
PsDscRunAsCredential_password: '{{ ansible_password }}'

SourceCredential_username: AdminUser
SourceCredential_password: PasswordForAdminUser
```

Note: On versions of Ansible older than 2.8, you should set `no_log: true` on the task definition in Ansible to ensure any credentials used are not stored in any log file or console output.

A `[PSCredential]` is defined with `EmbeddedInstance("MSFT_Credential")` in a DSC resource MOF definition.

CimInstance Type

A `[CimInstance]` object is used by DSC to store a dictionary object based on a custom class defined by that resource. Defining a value that takes in a `[CimInstance]` in YAML is the same as defining a dictionary in YAML. For example, to define a `[CimInstance]` value in Ansible:

```yaml
# [CimInstance]AuthenticationInfo == MSFT_xWebAuthenticationInformation
AuthenticationInfo:
  Anonymous: false
  Basic: true
  Digest: false
  Windows: true
```

In the above example, the CIM instance is a representation of the class MSFT_xWebAuthenticationInformation. This class accepts four boolean variables, `Anonymous`, `Basic`, `Digest`, and `Windows`. The keys to use in a `[CimInstance]` depend on the class it represents. Please read through the documentation of the resource to determine the keys that can be used and the types of each key value. The class definition is typically located in the `<resource name>.schema.mof`.

HashTable Type

A [HashTable] object is also a dictionary but does not have a strict set of keys that can/need to be defined. Like a [CimInstance], define it as a normal dictionary value in YAML. A [HashTable]] is defined with EmbeddedInstance("MSFT_KeyValuePair") in a DSC resource MOF definition.

Arrays

Simple type arrays like [string[]] or [UInt32[]] are defined as a list or as a comma-separated string which is then cast to their type. Using a list is recommended because the values are not manually parsed by the win_dsc module before being passed to the DSC engine. For example, to define a simple type array in Ansible:

```
# [string[]]
ValueData: entry1, entry2, entry3
ValueData:
- entry1
- entry2
- entry3

# [UInt32[]]
ReturnCode: 0,3010
ReturnCode:
- 0
- 3010
```

Complex type arrays like [CimInstance[]] (array of dicts), can be defined like this example:

```
# [CimInstance[]]BindingInfo == MSFT_xWebBindingInformation
BindingInfo:
- Protocol: https
  Port: 443
  CertificateStoreName: My
  CertificateThumbprint: C676A89018C4D5902353545343634F35E6B3A659
  HostName: DSCTest
  IPAddress: '*'
  SSLFlags: 1
- Protocol: http
  Port: 80
  IPAddress: '*'
```

The above example is an array with two values of the class MSFT_xWebBindingInformation. When defining a [CimInstance[]], be sure to read the resource documentation to find out what keys to use in the definition.

DateTime

A [DateTime] object is a DateTime string representing the date and time in the ISO 8601 date time format. The value for a [DateTime] field should be quoted in YAML to ensure the string is properly serialized to the Windows host. Here is an example of how to define a [DateTime] value in Ansible:

```yaml
# As UTC-0 (No timezone)
DateTime: '2019-02-22T13:57:31.2311892+00:00'

# As UTC+4
DateTime: '2019-02-22T17:57:31.2311892+04:00'

# As UTC-4
DateTime: '2019-02-22T09:57:31.2311892-04:00'
```

All the values above are equal to a UTC date time of February 22nd 2019 at 1:57pm with 31 seconds and 2311892 milliseconds.

Run As Another User

By default, DSC runs each resource as the SYSTEM account and not the account that Ansible uses to run the module. This means that resources that are dynamically loaded based on a user profile, like the HKEY_CURRENT_USER registry hive, will be loaded under the SYSTEM profile. The parameter PsDscRunAsCredential is a parameter that can be set for every DSC resource, and force the DSC engine to run under a different account. As PsDscRunAsCredential has a type of PSCredential, it is defined with the _username and _password suffix.

Using the Registry resource type as an example, this is how to define a task to access the HKEY_CURRENT_USER hive of the Ansible user:

```yaml
- name: Use win_dsc with PsDscRunAsCredential to run as a different user
  win_dsc:
    resource_name: Registry
    Ensure: Present
    Key: HKEY_CURRENT_USER\ExampleKey
    ValueName: TestValue
    ValueData: TestData
    PsDscRunAsCredential_username: '{{ ansible_user }}'
    PsDscRunAsCredential_password: '{{ ansible_password }}'
  no_log: true
```

Custom DSC Resources

DSC resources are not limited to the built-in options from Microsoft. Custom modules can be installed to manage other resources that are not usually available.

Finding Custom DSC Resources

You can use the PSGallery to find custom resources, along with documentation on how to install them on a Windows host.

The `Find-DscResource` cmdlet can also be used to find custom resources. For example:

```
# Find all DSC resources in the configured repositories
Find-DscResource

# Find all DSC resources that relate to SQL
Find-DscResource -ModuleName "*sql*"
```

Note: DSC resources developed by Microsoft that start with `x` means the resource is experimental and comes with no support.

Installing a Custom Resource

There are three ways that a DSC resource can be installed on a host:

- Manually with the `Install-Module` cmdlet
- Using the `win_psmodule` Ansible module
- Saving the module manually and copying it to another host

The following is an example of installing the `xWebAdministration` resources using `win_psmodule`:

```
- name: Install xWebAdministration DSC resource
  win_psmodule:
    name: xWebAdministration
    state: present
```

Once installed, the win_dsc module will be able to use the resource by referencing it with the `resource_name` option.

The first two methods above only work when the host has access to the internet. When a host does not have internet access, the module must first be installed using the methods above on another host with internet access and then copied across. To save a module to a local filepath, the following PowerShell cmdlet can be run:

```
Save-Module -Name xWebAdministration -Path C:\temp
```

This will create a folder called `xWebAdministration` in `C:\temp`, which can be copied to any host. For PowerShell to see this offline resource, it must be copied to a directory set in the `PSModulePath` environment variable. In most cases, the path `C:\Program Files\WindowsPowerShell\Module` is set through this variable, but the `win_path` module can be used to add different paths.

Examples

Extract a zip file

```
- name: Extract a zip file
  win_dsc:
    resource_name: Archive
    Destination: C:\temp\output
    Path: C:\temp\zip.zip
    Ensure: Present
```

Create a directory

```
- name: Create file with some text
  win_dsc:
    resource_name: File
    DestinationPath: C:\temp\file
    Contents: |
        Hello
        World
    Ensure: Present
    Type: File

- name: Create directory that is hidden is set with the System attribute
  win_dsc:
    resource_name: File
    DestinationPath: C:\temp\hidden-directory
    Attributes: Hidden,System
    Ensure: Present
    Type: Directory
```

Interact with Azure

```
- name: Install xAzure DSC resources
  win_psmodule:
    name: xAzure
    state: present

- name: Create virtual machine in Azure
  win_dsc:
    resource_name: xAzureVM
    ImageName: a699494373c04fc0bc8f2bb1389d6106__Windows-Server-2012-R2-201409.01-en.us-
↪127GB.vhd
    Name: DSCHOST01
    ServiceName: ServiceName
    StorageAccountName: StorageAccountName
    InstanceSize: Medium
    Windows: true
    Ensure: Present
```

(continues on next page)

(continued from previous page)

```
    Credential_username: '{{ ansible_user }}'
    Credential_password: '{{ ansible_password }}'
```

Setup IIS Website

```
- name: Install xWebAdministration module
  win_psmodule:
    name: xWebAdministration
    state: present

- name: Install IIS features that are required
  win_dsc:
    resource_name: WindowsFeature
    Name: '{{ item }}'
    Ensure: Present
  loop:
  - Web-Server
  - Web-Asp-Net45

- name: Setup web content
  win_dsc:
    resource_name: File
    DestinationPath: C:\inetpub\IISSite\index.html
    Type: File
    Contents: |
      <html>
      <head><title>IIS Site</title></head>
      <body>This is the body</body>
      </html>
    Ensure: present

- name: Create new website
  win_dsc:
    resource_name: xWebsite
    Name: NewIISSite
    State: Started
    PhysicalPath: C:\inetpub\IISSite\index.html
    BindingInfo:
    - Protocol: https
      Port: 8443
      CertificateStoreName: My
      CertificateThumbprint: C676A89018C4D5902353545343634F35E6B3A659
      HostName: DSCTest
      IPAddress: '*'
      SSLFlags: 1
    - Protocol: http
      Port: 8080
      IPAddress: '*'
    AuthenticationInfo:
      Anonymous: false
```

(continues on next page)

(continued from previous page)

```
      Basic: true
      Digest: false
      Windows: true
```

See also:

Ansible playbooks
    An introduction to playbooks

Ansible tips and tricks
    Tips and tricks for playbooks

List of Windows Modules
    Windows specific module list, all implemented in PowerShell

User Mailing List
    Have a question? Stop by the google group!

Real-time chat
    How to join Ansible chat channels

1.10.5 Windows performance

This document offers some performance optimizations you might like to apply to your Windows hosts to speed them up specifically in the context of using Ansible with them, and generally.

Optimize PowerShell performance to reduce Ansible task overhead

To speed up the startup of PowerShell by around 10x, run the following PowerShell snippet in an Administrator session. Expect it to take tens of seconds.

Note: If native images have already been created by the ngen task or service, you will observe no difference in performance (but this snippet will at that point execute faster than otherwise).

```
function Optimize-PowershellAssemblies {
  # NGEN powershell assembly, improves startup time of powershell by 10x
  $old_path = $env:path
  try {
    $env:path = [Runtime.InteropServices.RuntimeEnvironment]::GetRuntimeDirectory()
    [AppDomain]::CurrentDomain.GetAssemblies() | % {
      if (! $_.location) {continue}
      $Name = Split-Path $_.location -leaf
      if ($Name.startswith("Microsoft.PowerShell.")) {
        Write-Progress -Activity "Native Image Installation" -Status "$name"
        ngen install $_.location | % {"`t$_"}
      }
    }
  } finally {
    $env:path = $old_path
  }
}
Optimize-PowershellAssemblies
```

PowerShell is used by every Windows Ansible module. This optimization reduces the time PowerShell takes to start up, removing that overhead from every invocation.

This snippet uses the native image generator, ngen to pre-emptively create native images for the assemblies that PowerShell relies on.

Fix high-CPU-on-boot for VMs/cloud instances

If you are creating golden images to spawn instances from, you can avoid a disruptive high CPU task near startup through processing the ngen queue within your golden image creation, if you know the CPU types won't change between golden image build process and runtime.

Place the following near the end of your playbook, bearing in mind the factors that can cause native images to be invalidated (see MSDN).

```
- name: generate native .NET images for CPU
  win_dotnet_ngen:
```

1.10.6 Windows Frequently Asked Questions

Here are some commonly asked questions in regards to Ansible and Windows and their answers.

Note: This document covers questions about managing Microsoft Windows servers with Ansible. For questions about Ansible Core, please see the *general FAQ page*.

Does Ansible work with Windows XP or Server 2003?

Ansible does not work with Windows XP or Server 2003 hosts. Ansible does work with these Windows operating system versions:

- Windows Server 2008 [1]
- Windows Server 2008 R2 [1]
- Windows Server 2012
- Windows Server 2012 R2
- Windows Server 2016
- Windows Server 2019
- Windows 7 [1]
- Windows 8.1
- Windows 10

1 - See the *Server 2008 FAQ* entry for more details.

Ansible also has minimum PowerShell version requirements - please see *Setting up a Windows Host* for the latest information.

Are Server 2008, 2008 R2 and Windows 7 supported?

Microsoft ended Extended Support for these versions of Windows on January 14th, 2020, and Ansible deprecated official support in the 2.10 release. No new feature development will occur targeting these operating systems, and automated testing has ceased. However, existing modules and features will likely continue to work, and simple pull requests to resolve issues with these Windows versions may be accepted.

Can I manage Windows Nano Server with Ansible?

Ansible does not currently work with Windows Nano Server, since it does not have access to the full .NET Framework that is used by the majority of the modules and internal components.

Can Ansible run on Windows?

No, Ansible can only manage Windows hosts. Ansible cannot run on a Windows host natively, though it can run under the Windows Subsystem for Linux (WSL).

Note: The Windows Subsystem for Linux is not supported by Ansible and should not be used for production systems.

To install Ansible on WSL, the following commands can be run in the bash terminal:

```
sudo apt-get update
sudo apt-get install python3-pip git libffi-dev libssl-dev -y
pip install --user ansible pywinrm
```

To run Ansible from source instead of a release on the WSL, simply uninstall the pip installed version and then clone the git repo.

```
pip uninstall ansible -y
git clone https://github.com/ansible/ansible.git
source ansible/hacking/env-setup

# To enable Ansible on login, run the following
echo ". ~/ansible/hacking/env-setup -q' >> ~/.bashrc
```

If you encounter timeout errors when running Ansible on the WSL, this may be due to an issue with `sleep` not returning correctly. The following workaround may resolve the issue:

```
mv /usr/bin/sleep /usr/bin/sleep.orig
ln -s /bin/true /usr/bin/sleep
```

Another option is to use WSL 2 if running Windows 10 later than build 2004.

```
wsl --set-default-version 2
```

Can I use SSH keys to authenticate to Windows hosts?

You cannot use SSH keys with the WinRM or PSRP connection plugins. These connection plugins use X509 certificates for authentication instead of the SSH key pairs that SSH uses.

The way X509 certificates are generated and mapped to a user is different from the SSH implementation; consult the *Windows Remote Management* documentation for more information.

Ansible 2.8 has added an experimental option to use the SSH connection plugin, which uses SSH keys for authentication, for Windows servers. See *this question* for more information.

Why can I run a command locally that does not work under Ansible?

Ansible executes commands through WinRM. These processes are different from running a command locally in these ways:

- Unless using an authentication option like CredSSP or Kerberos with credential delegation, the WinRM process does not have the ability to delegate the user's credentials to a network resource, causing `Access is Denied` errors.
- All processes run under WinRM are in a non-interactive session. Applications that require an interactive session will not work.
- When running through WinRM, Windows restricts access to internal Windows APIs like the Windows Update API and DPAPI, which some installers and programs rely on.

Some ways to bypass these restrictions are to:

- Use `become`, which runs a command as it would when run locally. This will bypass most WinRM restrictions, as Windows is unaware the process is running under WinRM when `become` is used. See the *Understanding privilege escalation: become* documentation for more information.
- Use a scheduled task, which can be created with `win_scheduled_task`. Like `become`, it will bypass all WinRM restrictions, but it can only be used to run commands, not modules.
- Use `win_psexec` to run a command on the host. PSExec does not use WinRM and so will bypass any of the restrictions.
- To access network resources without any of these workarounds, you can use CredSSP or Kerberos with credential delegation enabled.

See *Understanding privilege escalation: become* more info on how to use become. The limitations section at *Windows Remote Management* has more details around WinRM limitations.

This program won't install on Windows with Ansible

See *this question* for more information about WinRM limitations.

What Windows modules are available?

Most of the Ansible modules in Ansible Core are written for a combination of Linux/Unix machines and arbitrary web services. These modules are written in Python and most of them do not work on Windows.

Because of this, there are dedicated Windows modules that are written in PowerShell and are meant to be run on Windows hosts. A list of these modules can be found here.

In addition, the following Ansible Core modules/action-plugins work with Windows:

- add_host
- assert
- async_status
- debug
- fail
- fetch
- group_by
- include
- include_role
- include_vars
- meta
- pause
- raw
- script
- set_fact
- set_stats
- setup
- slurp
- template (also: win_template)
- wait_for_connection

Ansible Windows modules exist in the Ansible.Windows, Community.Windows, and Chocolatey.Chocolatey collections.

Can I run Python modules on Windows hosts?

No, the WinRM connection protocol is set to use PowerShell modules, so Python modules will not work. A way to bypass this issue to use `delegate_to:  localhost` to run a Python module on the Ansible controller. This is useful if during a playbook, an external service needs to be contacted and there is no equivalent Windows module available.

Can I connect to Windows hosts over SSH?

Ansible 2.8 has added an experimental option to use the SSH connection plugin to manage Windows hosts. To connect to Windows hosts over SSH, you must install and configure the Win32-OpenSSH fork that is in development with Microsoft on the Windows host(s). While most of the basics should work with SSH, `Win32-OpenSSH` is rapidly changing, with new features added and bugs fixed in every release. It is highly recommend you install the latest release of `Win32-OpenSSH` from the GitHub Releases page when using it with Ansible on Windows hosts.

To use SSH as the connection to a Windows host, set the following variables in the inventory:

```
ansible_connection=ssh

# Set either cmd or powershell not both
ansible_shell_type=cmd
# ansible_shell_type=powershell
```

The value for `ansible_shell_type` should either be `cmd` or `powershell`. Use `cmd` if the `DefaultShell` has not been configured on the SSH service and `powershell` if that has been set as the `DefaultShell`.

Why is connecting to a Windows host through SSH failing?

Unless you are using `Win32-OpenSSH` as described above, you must connect to Windows hosts using *Windows Remote Management*. If your Ansible output indicates that SSH was used, either you did not set the connection vars properly or the host is not inheriting them correctly.

Make sure `ansible_connection: winrm` is set in the inventory for the Windows host(s).

Why are my credentials being rejected?

This can be due to a myriad of reasons unrelated to incorrect credentials.

See HTTP 401/Credentials Rejected at *Setting up a Windows Host* for a more detailed guide of this could mean.

Why am I getting an error SSL CERTIFICATE_VERIFY_FAILED?

When the Ansible controller is running on Python 2.7.9+ or an older version of Python that has backported SSLContext (like Python 2.7.5 on RHEL 7), the controller will attempt to validate the certificate WinRM is using for an HTTPS connection. If the certificate cannot be validated (such as in the case of a self signed cert), it will fail the verification process.

To ignore certificate validation, add `ansible_winrm_server_cert_validation: ignore` to inventory for the Windows host.

See also:

Windows Guides
The Windows documentation index

Ansible playbooks
An introduction to playbooks

Ansible tips and tricks
Tips and tricks for playbooks

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

1.10.7 Managing BSD hosts with Ansible

Managing BSD machines is different from managing other Unix-like machines. If you have managed nodes running BSD, review these topics.

Connecting to BSD nodes

Ansible connects to managed nodes using OpenSSH by default. This works on BSD if you use SSH keys for authentication. However, if you use SSH passwords for authentication, Ansible relies on sshpass. Most versions of sshpass do not deal well with BSD login prompts, so when using SSH passwords against BSD machines, use `paramiko` to connect instead of OpenSSH. You can do this in ansible.cfg globally or you can set it as an inventory/group/host variable. For example:

```
[freebsd]
mybsdhost1 ansible_connection=paramiko
```

Bootstrapping BSD

Ansible is agentless by default, however, it requires Python on managed nodes. Only the raw module will operate without Python. Although this module can be used to bootstrap Ansible and install Python on BSD variants (see below), it is very limited and the use of Python is required to make full use of Ansible's features.

The following example installs Python which includes the json library required for full functionality of Ansible. On your control machine you can execute the following for most versions of FreeBSD:

```
ansible -m raw -a "pkg install -y python" mybsdhost1
```

Or for OpenBSD:

```
ansible -m raw -a "pkg_add python%3.8"
```

Once this is done you can now use other Ansible modules apart from the raw module.

Note: This example demonstrated using pkg on FreeBSD and pkg_add on OpenBSD, however you should be able to substitute the appropriate package tool for your BSD; the package name may also differ. Refer to the package list or documentation of the BSD variant you are using for the exact Python package name you intend to install.

Setting the Python interpreter

To support a variety of Unix-like operating systems and distributions, Ansible cannot always rely on the existing environment or env variables to locate the correct Python binary. By default, modules point at /usr/bin/python as this is the most common location. On BSD variants, this path may differ, so it is advised to inform Ansible of the binary's location. See *INTERPRETER_PYTHON*. For example, set ansible_python_interpreter inventory variable:

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python
[openbsd:vars]
ansible_python_interpreter=/usr/local/bin/python3.8
```

FreeBSD packages and ports

In FreeBSD, there is no guarantee that either /usr/local/bin/python executable file or a link to an executable file is installed by default. The best practice for a remote host, with respect to Ansible, is to install at least the Python version supported by Ansible, for example, lang/python38, and both meta ports lang/python3 and lang/python. Quoting from */usr/ports/lang/python3/pkg-descr*:

```
This is a meta port to the Python 3.x interpreter and provides symbolic links
to bin/python3, bin/pydoc3, bin/idle3 and so on to allow compatibility with
minor version agnostic python scripts.
```

Quoting from */usr/ports/lang/python/pkg-descr*:

```
This is a meta port to the Python interpreter and provides symbolic links
to bin/python, bin/pydoc, bin/idle and so on to allow compatibility with
version agnostic python scripts.
```

As a result, the following packages are installed:

```
shell> pkg info | grep python
python-3.8_3,2                 "meta-port" for the default version of Python interpreter
python3-3_3                    Meta-port for the Python interpreter 3.x
python38-3.8.12_1              Interpreted object-oriented programming language
```

and the following executables and links

```
shell> ll /usr/local/bin/ | grep python
lrwxr-xr-x  1 root  wheel      7 Jan 24 08:30 python@ -> python3
lrwxr-xr-x  1 root  wheel     14 Jan 24 08:30 python-config@ -> python3-config
lrwxr-xr-x  1 root  wheel      9 Jan 24 08:29 python3@ -> python3.8
lrwxr-xr-x  1 root  wheel     16 Jan 24 08:29 python3-config@ -> python3.8-config
-r-xr-xr-x  1 root  wheel   5248 Jan 13 01:12 python3.8*
-r-xr-xr-x  1 root  wheel   3153 Jan 13 01:12 python3.8-config*
```

INTERPRETER_PYTHON_FALLBACK

Since version 2.8 Ansible provides a useful variable `ansible_interpreter_python_fallback` to specify a list of paths to search for Python. See *INTERPRETER_PYTHON_FALLBACK*. This list will be searched and the first item found will be used. For example, the configuration below would make the installation of the meta-ports in the previous section redundant, that is, if you don't install the Python meta ports the first two items in the list will be skipped and `/usr/local/bin/python3.8` will be discovered.

```
ansible_interpreter_python_fallback=['/usr/local/bin/python', '/usr/local/bin/python3',
↪'/usr/local/bin/python3.8']
```

You can use this variable, prolonged by the lower versions of Python, and put it, for example, into the `group_vars/all`. Then, override it for specific groups in `group_vars/{group1, group2, ...}` and for specific hosts in `host_vars/{host1, host2, ...}` if needed. See *Variable precedence: Where should I put a variable?*.

Debug the discovery of Python

For example, given the inventory

```
shell> cat hosts
[test]
test_11
test_12
test_13

[test:vars]
ansible_connection=ssh
ansible_user=admin
ansible_become=true
ansible_become_user=root
ansible_become_method=sudo
ansible_interpreter_python_fallback=['/usr/local/bin/python', '/usr/local/bin/python3',
↪'/usr/local/bin/python3.8']
ansible_perl_interpreter=/usr/local/bin/perl
```

The playbook below

```
shell> cat playbook.yml
- hosts: test_11
  gather_facts: false
  tasks:
    - command: which python
      register: result
    - debug:
        var: result.stdout
    - debug:
        msg: |-
          {% for i in _vars %}
          {{ i }}:
            {{ lookup('vars', i)|to_nice_yaml|indent(2) }}
          {% endfor %}
      vars:
        _vars: "{{ query('varnames', '.*python.*') }}"
```

displays the details

```
shell> ansible-playbook -i hosts playbook.yml

PLAY [test_11]␣
↪*****************************************************************************

TASK [command]␣
↪*****************************************************************************
[WARNING]: Platform freebsd on host test_11 is using the discovered Python interpreter at
/usr/local/bin/python, but future installation of another Python interpreter could␣
↪change the
meaning of that path. See https://docs.ansible.com/ansible-
core/2.12/reference_appendices/interpreter_discovery.html for more information.
changed: [test_11]

TASK [debug]␣
↪*******************************************************************************
ok: [test_11] =>
  result.stdout: /usr/local/bin/python

TASK [debug]␣
↪*******************************************************************************
ok: [test_11] =>
  msg: |-
    ansible_interpreter_python_fallback:
      - /usr/local/bin/python
      - /usr/local/bin/python3
      - /usr/local/bin/python3.8

    discovered_interpreter_python:
      /usr/local/bin/python

    ansible_playbook_python:
      /usr/bin/python3
```

You can see that the first item from the list `ansible_interpreter_python_fallback` was discovered at the FreeBSD remote host. The variable `ansible_playbook_python` keeps the path to Python at the Linux controller that ran the playbook.

Regarding the warning, quoting from *INTERPRETER_PYTHON*

```
The fallback behavior will issue a warning that the interpreter
should be set explicitly (since interpreters installed later may
change which one is used). This warning behavior can be disabled by
setting auto_silent or auto_legacy_silent. ...
```

You can either ignore it or get rid of it by setting the variable `ansible_python_interpreter=auto_silent` because this is, actually, what you want by using `/usr/local/bin/python` (*"interpreters installed later may change which one is used"*). For example

```
shell> cat hosts
[test]
test_11
```

(continues on next page)

(continued from previous page)

```
test_12
test_13

[test:vars]
ansible_connection=ssh
ansible_user=admin
ansible_become=true
ansible_become_user=root
ansible_become_method=sudo
ansible_interpreter_python_fallback=['/usr/local/bin/python', '/usr/local/bin/python3',
↪'/usr/local/bin/python3.8']
ansible_python_interpreter=auto_silent
ansible_perl_interpreter=/usr/local/bin/perl
```

See also:

- *Interpreter Discovery*
- FreeBSD Wiki: Ports/DEFAULT_VERSIONS

Additional variables

If you use additional plugins beyond those bundled with Ansible, you can set similar variables for `bash`, `perl` or `ruby`, depending on how the plugin is written. For example:

```
[freebsd:vars]
ansible_python_interpreter=/usr/local/bin/python
ansible_perl_interpreter=/usr/local/bin/perl
```

Which modules are available?

The majority of the core Ansible modules are written for a combination of Unix-like machines and other generic services, so most should function well on the BSDs with the obvious exception of those that are aimed at Linux-only technologies (such as LVG).

Using BSD as the control node

Using BSD as the control machine is as simple as installing the Ansible package for your BSD variant or by following the `pip` or 'from source' instructions.

BSD facts

Ansible gathers facts from the BSDs in a similar manner to Linux machines, but since the data, names and structures can vary for network, disks and other devices, one should expect the output to be slightly different yet still familiar to a BSD administrator.

BSD efforts and contributions

BSD support is important to us at Ansible. Even though the majority of our contributors use and target Linux we have an active BSD community and strive to be as BSD-friendly as possible. Please feel free to report any issues or incompatibilities you discover with BSD; pull requests with an included fix are also welcome!

See also:

Introduction to ad hoc commands
    Examples of basic commands

Working with playbooks
    Learning ansible's configuration management language

Should you develop a module?
    How to write modules

Mailing List
    Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
    How to join Ansible chat channels

1.11 Ansible tips and tricks

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible tips and tricks guide. These tips and tricks have helped us optimize our Ansible usage and we offer them here as suggestions. We hope they will help you organize content, write playbooks, maintain inventory, and execute Ansible. Ultimately, though, you should use Ansible in the way that makes most sense for your organization and your goals.

1.11.1 General tips

These concepts apply to all Ansible activities and artifacts.

Keep it simple

Whenever you can, do things simply. Use advanced features only when necessary, and select the feature that best matches your use case. For example, you will probably not need `vars`, `vars_files`, `vars_prompt` and `--extra-vars` all at once, while also using an external inventory file. If something feels complicated, it probably is. Take the time to look for a simpler solution.

Use version control

Keep your playbooks, roles, inventory, and variables files in git or another version control system and make commits to the repository when you make changes. Version control gives you an audit trail describing when and why you changed the rules that automate your infrastructure.

Customize the CLI output

You can change the output from Ansible CLI commands using *Callback plugins*.

1.11.2 Playbook tips

These tips help make playbooks and roles easier to read, maintain, and debug.

Use whitespace

Generous use of whitespace, for example, a blank line before each block or task, makes a playbook easy to scan.

Always name tasks

Task names are optional, but extremely useful. In its output, Ansible shows you the name of each task it runs. Choose names that describe what each task does and why.

Always mention the state

For many modules, the 'state' parameter is optional. Different modules have different default settings for 'state', and some modules support several 'state' settings. Explicitly setting 'state=present' or 'state=absent' makes playbooks and roles clearer.

Use comments

Even with task names and explicit state, sometimes a part of a playbook or role (or inventory/variable file) needs more explanation. Adding a comment (any line starting with '#') helps others (and possibly yourself in future) understand what a play or task (or variable setting) does, how it does it, and why.

1.11.3 Inventory tips

These tips help keep your inventory well organized.

Use dynamic inventory with clouds

With cloud providers and other systems that maintain canonical lists of your infrastructure, use *dynamic inventory* to retrieve those lists instead of manually updating static inventory files. With cloud resources, you can use tags to differentiate production and staging environments.

Group inventory by function

A system can be in multiple groups. See *How to build your inventory* and *Patterns: targeting hosts and groups*. If you create groups named for the function of the nodes in the group, for example *webservers* or *dbservers*, your playbooks can target machines based on function. You can assign function-specific variables using the group variable system, and design Ansible roles to handle function-specific use cases. See *Roles*.

Separate production and staging inventory

You can keep your production environment separate from development, test, and staging environments by using separate inventory files or directories for each environment. This way you pick with -i what you are targeting. Keeping all your environments in one file can lead to surprises! For example, all vault passwords used in an inventory need to be available when using that inventory. If an inventory contains both production and development environments, developers using that inventory would be able to access production secrets.

Keep vaulted variables safely visible

You should encrypt sensitive or secret variables with Ansible Vault. However, encrypting the variable names as well as the variable values makes it hard to find the source of the values. To circumvent this, you can encrypt the variables individually using `ansible-vault encrypt_string`, or add the following layer of indirection to keep the names of your variables accessible (by `grep`, for example) without exposing any secrets:

1. Create a `group_vars/` subdirectory named after the group.
2. Inside this subdirectory, create two files named `vars` and `vault`.
3. In the `vars` file, define all of the variables needed, including any sensitive ones.
4. Copy all of the sensitive variables over to the `vault` file and prefix these variables with `vault_`.
5. Adjust the variables in the `vars` file to point to the matching `vault_` variables using jinja2 syntax: `db_password: {{ vault_db_password }}`.
6. Encrypt the `vault` file to protect its contents.
7. Use the variable name from the `vars` file in your playbooks.

When running a playbook, Ansible finds the variables in the unencrypted file, which pulls the sensitive variable values from the encrypted file. There is no limit to the number of variable and vault files or their names.

Note that using this strategy in your inventory still requires *all vault passwords to be available* (for example for `ansible-playbook` or AWX/Ansible Tower) when run with that inventory.

1.11.4 Execution tricks

These tips apply to using Ansible, rather than to Ansible artifacts.

Try it in staging first

Testing changes in a staging environment before rolling them out in production is always a great idea. Your environments need not be the same size and you can use group variables to control the differences between those environments.

Update in batches

Use the 'serial' keyword to control how many machines you update at once in the batch. See *Controlling where tasks run: delegation and local actions*.

Handling OS and distro differences

Group variables files and the `group_by` module work together to help Ansible execute across a range of operating systems and distributions that require different settings, packages, and tools. The `group_by` module creates a dynamic group of hosts that match certain criteria. This group does not need to be defined in the inventory file. This approach lets you execute different tasks on different operating systems or distributions.

For example, the following play categorizes all systems into dynamic groups based on the operating system name:

```yaml
- name: talk to all hosts just so we can learn about them
  hosts: all
  tasks:
    - name: Classify hosts depending on their OS distribution
      group_by:
        key: os_{{ ansible_facts['distribution'] }}
```

Subsequent plays can use these groups as patterns on the `hosts` line as follows:

```yaml
- hosts: os_CentOS
  gather_facts: False
  tasks:
    # Tasks for CentOS hosts only go in this play.
    - name: Ping my CentOS hosts
      ansible.builtin.ping:
```

You can also add group-specific settings in group vars files. In the following example, CentOS machines get the value of '42' for *asdf* but other machines get '10'. You can also use group vars files to apply roles to systems as well as set variables.

```yaml
---
# file: group_vars/all
asdf: 10

---
# file: group_vars/os_CentOS.yml
asdf: 42
```

Note: All three names must match: the name created by the `group_by` task, the name of the pattern in subsequent plays, and the name of the group vars file.

You can use the same setup with `include_vars` when you only need OS-specific variables, not tasks:

```
- hosts: all
  tasks:
    - name: Set OS distribution dependent variables
      include_vars: "os_{{ ansible_facts['distribution'] }}.yml"
    - debug:
        var: asdf
```

This pulls in variables from the *group_vars/os_CentOS.yml* file.

See also:

YAML Syntax
: Learn about YAML syntax

Working with playbooks
: Review the basic playbook features

Collection Index
: Browse existing collections, modules, and plugins

Should you develop a module?
: Learn how to extend Ansible by writing your own modules

Patterns: targeting hosts and groups
: Learn about how to select hosts

GitHub examples directory
: Complete playbook files from the github project source

Mailing List
: Questions? Help? Ideas? Stop by the list on Google Groups

1.11.5 Sample Ansible setup

You have learned about playbooks, inventory, roles, and variables. This section combines all those elements, and outlines a sample setup for automating a web service. You can find more example playbooks that illustrate these patterns in our ansible-examples repository. (NOTE: These examples do not use all of the latest features, but are still an excellent reference.).

The sample setup organizes playbooks, roles, inventory, and files with variables by function. Tags at the play and task level provide greater granularity and control. This is a powerful and flexible approach, but there are other ways to organize Ansible content. Your usage of Ansible should fit your needs, so feel free to modify this approach and organize your content accordingly.

Sample directory layout

This layout organizes most tasks in roles, with a single inventory file for each environment and a few playbooks in the top-level directory:

```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1.yml             # here we assign variables to particular groups
   group2.yml
host_vars/
   hostname1.yml          # here we assign variables to particular systems
   hostname2.yml

library/                  # if any custom modules, put them here (optional)
module_utils/             # if any custom module_utils to support modules, put them here␣
↪(optional)
filter_plugins/           # if any custom filter plugins, put them here (optional)

site.yml                  # main playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier
tasks/                    # task files included from playbooks
    webservers-extra.yml  # <-- avoids confusing playbook with task files
```

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies and optional Galaxy info
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
```

(continues on next page)

(continued from previous page)

```
        lookup_plugins/    # or other types of plugins, like lookup in this case

    webtier/               # same kind of structure as "common" was above, done for the␣
↪webtier role
    monitoring/            # ""
    fooapp/                # ""
```

Note: By default, Ansible assumes your playbooks are stored in one directory with roles stored in a sub-directory called `roles/`. With more tasks to automate, you can consider moving your playbooks into a sub-directory called `playbooks/`. If you do this, you must configure the path to your `roles/` directory using the `roles_path` setting in the `ansible.cfg` file.

Alternative directory layout

You can also put each inventory file with its `group_vars/host_vars` in a separate directory. This is particularly useful if your `group_vars/host_vars` do not have that much in common in different environments. The layout could look like this example:

```
inventories/
   production/
      hosts               # inventory file for production servers
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         hostname1.yml    # here we assign variables to particular systems
         hostname2.yml

   staging/
      hosts               # inventory file for staging environment
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         stagehost1.yml   # here we assign variables to particular systems
         stagehost2.yml

library/
module_utils/
filter_plugins/

site.yml
webservers.yml
dbservers.yml

roles/
    common/
    webtier/
    monitoring/
    fooapp/
```

This layout gives you more flexibility for larger environments, as well as a total separation of inventory variables between different environments. However, this approach is harder to maintain, because there are more files. For more information on organizing group and host variables, see *Organizing host and group variables*.

Sample group and host variables

These sample group and host files with variables contain the values that apply to each machine or a group of machines. For instance, the data center in Atlanta has its own NTP servers. As a result, when setting up the `ntp.conf` file, you could use similar code as in this example:

```
---
# file: group_vars/atlanta
ntp: ntp-atlanta.example.com
backup: backup-atlanta.example.com
```

Similarly, hosts in the webservers group have some configuration that does not apply to the database servers:

```
---
# file: group_vars/webservers
apacheMaxRequestsPerChild: 3000
apacheMaxClients: 900
```

Default values, or values that are universally true, belong in a file called `group_vars/all`:

```
---
# file: group_vars/all
ntp: ntp-boston.example.com
backup: backup-boston.example.com
```

If necessary, you can define specific hardware variance in systems in the `host_vars` directory:

```
---
# file: host_vars/db-bos-1.example.com
foo_agent_port: 86
bar_agent_port: 99
```

If you use *dynamic inventory*, Ansible creates many dynamic groups automatically. As a result, a tag like `class:webserver` will load in variables from the file `group_vars/ec2_tag_class_webserver` automatically.

Note: You can access host variables with a special variable called `hostvars`. See *Special Variables* for a list of these variables. The `hostvars` variable can access only host-specific variables, not group variables.

Sample playbooks organized by function

With this setup, a single playbook can define the entire infrastructure. The `site.yml` playbook imports two other playbooks. One for the webservers and one for the database servers:

```
---
# file: site.yml
- import_playbook: webservers.yml
- import_playbook: dbservers.yml
```

The webservers.yml playbook, also at the top level, maps the configuration of the webservers group to the roles related to the webservers group:

```
---
# file: webservers.yml
- hosts: webservers
  roles:
    - common
    - webtier
```

With this setup, you can configure your entire infrastructure by running site.yml. Alternatively, to configure just a portion of your infrastructure, run webservers.yml. This is similar to the Ansible --limit parameter but a little more explicit:

```
ansible-playbook site.yml --limit webservers
ansible-playbook webservers.yml
```

Sample task and handler files in a function-based role

Ansible loads any file called main.yml in a role sub-directory. This sample tasks/main.yml file configures NTP:

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum:
    name: ntp
    state: present
  tags: ntp

- name: be sure ntp is configured
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp

- name: be sure ntpd is running and enabled
  ansible.builtin.service:
    name: ntpd
    state: started
    enabled: true
  tags: ntp
```

Here is an example handlers file. Handlers are only triggered when certain tasks report changes. Handlers run at the end of each play:

```
---
# file: roles/common/handlers/main.yml
- name: restart ntpd
  ansible.builtin.service:
```

(continues on next page)

(continued from previous page)

```
    name: ntpd
    state: restarted
```

See *Roles* for more information.

What the sample setup enables

The basic organizational structure described above enables a lot of different automation options. To reconfigure your entire infrastructure:

```
ansible-playbook -i production site.yml
```

To reconfigure NTP on everything:

```
ansible-playbook -i production site.yml --tags ntp
```

To reconfigure only the webservers:

```
ansible-playbook -i production webservers.yml
```

To reconfigure only the webservers in Boston:

```
ansible-playbook -i production webservers.yml --limit boston
```

To reconfigure only the first 10 webservers in Boston, and then the next 10:

```
ansible-playbook -i production webservers.yml --limit boston[0:9]
ansible-playbook -i production webservers.yml --limit boston[10:19]
```

The sample setup also supports basic ad hoc commands:

```
ansible boston -i production -m ping
ansible boston -i production -m command -a '/sbin/reboot'
```

To discover what tasks would run or what hostnames would be affected by a particular Ansible command:

```
# confirm what task names would be run if I ran this command and said "just␣
↪ntp tasks"
ansible-playbook -i production webservers.yml --tags ntp --list-tasks

# confirm what hostnames might be communicated with if I said "limit to boston"
ansible-playbook -i production webservers.yml --limit boston --list-hosts
```

Organizing for deployment or configuration

The sample setup illustrates a typical configuration topology. When you do multi-tier deployments, you will likely need some additional playbooks that hop between tiers to roll out an application. In this case, you can augment `site.yml` with playbooks like `deploy_exampledotcom.yml`. However, the general concepts still apply. With Ansible you can deploy and configure using the same utility. Therefore, you will probably reuse groups and keep the OS configuration in separate playbooks or roles from the application deployment.

Consider "playbooks" as a sports metaphor – you can have one set of plays to use against all your infrastructure. Then you have situational plays that you use at different times and for different purposes.

Using local Ansible modules

If a playbook has a `./library` directory relative to its YAML file, you can use this directory to add Ansible modules automatically to the module path. This organizes modules with playbooks. For example, see the directory structure at the start of this section.

See also:

YAML Syntax
Learn about YAML syntax

Working with playbooks
Review the basic playbook features

Collection Index
Browse existing collections, modules, and plugins

Should you develop a module?
Learn how to extend Ansible by writing your own modules

Patterns: targeting hosts and groups
Learn about how to select hosts

GitHub examples directory
Complete playbook files from the github project source

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

1.12 Ansible Community Guide

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible Community Guide!

The purpose of this guide is to teach you everything you need to know about being a contributing member of the Ansible community. All types of contributions are welcome and necessary for Ansible's continued success.

1.12.1 Getting started

Welcome and thank you for getting more involved with the Ansible community. Here are some ways you can get started.

Community Code of Conduct

Topics

Every community can be strengthened by a diverse variety of viewpoints, insights, opinions, skillsets, and skill levels. However, with diversity comes the potential for disagreement and miscommunication. The purpose of this Code of Conduct is to ensure that disagreements and differences of opinion are conducted respectfully and on their own merits, without personal attacks or other behavior that might create an unsafe or unwelcoming environment.

These policies are not designed to be a comprehensive set of Things You Cannot Do. We ask that you treat your fellow community members with respect and courtesy, and in general, Don't Be A Jerk. This Code of Conduct is meant to be followed in spirit as much as in letter and is not exhaustive.

All Ansible events and participants therein are governed by this Code of Conduct and anti-harassment policy. We expect organizers to enforce these guidelines throughout all events, and we expect attendees, speakers, sponsors, and volunteers to help ensure a safe environment for our whole community. Specifically, this Code of Conduct covers participation in all Ansible-related forums and mailing lists, code and documentation contributions, public chat (Matrix, IRC), private correspondence, and public meetings.

Ansible community members are…

Considerate

Contributions of every kind have far-ranging consequences. Just as your work depends on the work of others, decisions you make surrounding your contributions to the Ansible community will affect your fellow community members. You are strongly encouraged to take those consequences into account while making decisions.

Patient

Asynchronous communication can come with its own frustrations, even in the most responsive of communities. Please remember that our community is largely built on volunteered time, and that questions, contributions, and requests for support may take some time to receive a response. Repeated "bumps" or "reminders" in rapid succession are not good displays of patience. Additionally, it is considered poor manners to ping a specific person with general questions. Pose your question to the community as a whole, and wait patiently for a response.

Respectful

Every community inevitably has disagreements, but remember that it is possible to disagree respectfully and courteously. Disagreements are never an excuse for rudeness, hostility, threatening behavior, abuse (verbal or physical), or personal attacks.

Kind

Everyone should feel welcome in the Ansible community, regardless of their background. Please be courteous, respectful and polite to fellow community members. Do not make or post offensive comments related to skill level, gender, gender identity or expression, sexual orientation, disability, physical appearance, body size, race, or religion. Sexualized images or imagery, real or implied violence, intimidation, oppression, stalking, sustained disruption of activities, publishing the personal information of others without explicit permission to do so, unwanted physical contact, and unwelcome sexual attention are all strictly prohibited. Additionally, you are encouraged not to make assumptions about the background or identity of your fellow community members.

Inquisitive

The only stupid question is the one that does not get asked. We encourage our users to ask early and ask often. Rather than asking whether you can ask a question (the answer is always yes!), instead, simply ask your question. You are encouraged to provide as many specifics as possible. Code snippets in the form of Gists or other paste site links are almost always needed in order to get the most helpful answers. Refrain from pasting multiple lines of code directly into the chat channels - instead use gist.github.com or another paste site to provide code snippets.

Helpful

The Ansible community is committed to being a welcoming environment for all users, regardless of skill level. We were all beginners once upon a time, and our community cannot grow without an environment where new users feel safe and comfortable asking questions. It can become frustrating to answer the same questions repeatedly; however, community members are expected to remain courteous and helpful to all users equally, regardless of skill or knowledge level. Avoid providing responses that prioritize snideness and snark over useful information. At the same time, everyone is expected to read the provided documentation thoroughly. We are happy to answer questions, provide strategic guidance, and suggest effective workflows, but we are not here to do your job for you.

Anti-harassment policy

Harassment includes (but is not limited to) all of the following behaviors:

- Offensive comments related to gender (including gender expression and identity), age, sexual orientation, disability, physical appearance, body size, race, and religion
- Derogatory terminology including words commonly known to be slurs
- Posting sexualized images or imagery in public spaces
- Deliberate intimidation
- Stalking
- Posting others' personal information without explicit permission
- Sustained disruption of talks or other events
- Inappropriate physical contact
- Unwelcome sexual attention

Participants asked to stop any harassing behavior are expected to comply immediately. Sponsors are also subject to the anti-harassment policy. In particular, sponsors should not use sexualized images, activities, or other material. Meetup organizing staff and other volunteer organizers should not use sexualized attire or otherwise create a sexualized environment at community events.

In addition to the behaviors outlined above, continuing to behave a certain way after you have been asked to stop also constitutes harassment, even if that behavior is not specifically outlined in this policy. It is considerate and respectful to stop doing something after you have been asked to stop, and all community members are expected to comply with such requests immediately.

Policy violations

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting codeofconduct@ansible.com, to anyone with administrative power in community chat (Admins or Moderators on Matrix, ops on IRC), or to the local organizers of an event. Meetup organizers are encouraged to prominently display points of contact for reporting unacceptable behavior at local events.

If a participant engages in harassing behavior, the meetup organizers may take any action they deem appropriate. These actions may include but are not limited to warning the offender, expelling the offender from the event, and barring the offender from future community events.

Organizers will be happy to help participants contact security or local law enforcement, provide escorts to an alternate location, or otherwise assist those experiencing harassment to feel safe for the duration of the meetup. We value the safety and well-being of our community members and want everyone to feel welcome at our events, both online and offline.

We expect all participants, organizers, speakers, and attendees to follow these policies at all of our event venues and event-related social events.

The Ansible Community Code of Conduct is licensed under the Creative Commons Attribution-Share Alike 3.0 license. Our Code of Conduct was adapted from Codes of Conduct of other open source projects, including:

- Contributor Covenant
- Elastic
- The Fedora Project
- OpenStack
- Puppet Labs
- Ubuntu

Contributors License Agreement

By contributing you agree that these contributions are your own (or approved by your employer) and you grant a full, complete, irrevocable copyright license to all users and developers of the project, present and future, pursuant to the license of the project.

Communicating with the Ansible community

- *Ansible Community Topics*
- *Ansible Automation Platform support questions*
- *The Bullhorn*

Code of Conduct

All communication and interactions in the Ansible Community are governed by our *Community Code of Conduct*. Please read and understand it!

Asking questions over email

If you want to keep up with Ansible news, need help, or have a question, you can use one of the Ansible mailing lists. Each list covers a particular topic. Read the descriptions here to find the best list for your question.

Your first post to the mailing list will be moderated (to reduce spam), so please allow up to a day or so for your first post to appear.

- Ansible Announce list is a read-only list that shares information about new releases of Ansible, and also rare infrequent event information, such as announcements about an upcoming AnsibleFest, which is our official conference series. Worth subscribing to!
- Ansible AWX List is for Ansible AWX
- Ansible Development List is for questions about developing Ansible modules (mostly in Python), fixing bugs in the Ansible core code, asking about prospective feature design, or discussions about extending Ansible or features in progress.
- Ansible Outreach List help with promoting Ansible and Ansible Meetups
- Ansible Project List is for sharing Ansible tips, answering questions about playbooks and roles, and general user discussion.
- Molecule Discussions is designed to aid with the development and testing of Ansible roles with Molecule.

The Ansible mailing lists are hosted on Google, but you do not need a Google account to subscribe. To subscribe to a group from a non-Google account, send an email to the subscription address requesting the subscription. For example: `ansible-devel+subscribe@googlegroups.com`.

Real-time chat

For real-time interactions, conversations in the Ansible community happen over two chat protocols: Matrix and IRC. We maintain a bridge between Matrix and IRC, so you can choose whichever protocol you prefer. All channels exist in both places. Join a channel any time to ask questions, participate in a Working Group meeting, or just say hello.

Ansible community on Matrix

To join the community using Matrix, you need two things:

- a Matrix account (from Matrix.org or any other Matrix homeserver)
- a Matrix client (we recommend Element Webchat)

The Ansible community maintains its own Matrix homeserver at `ansible.im`, however public registration is currently unavailable.

Matrix chat supports:

- persistence (when you log on, you see all messages since you last logged off)
- edits (Lets you fix typos and so on. **NOTE** Each edit you make on Matrix re-sends the message to IRC. Please try to avoid multiple edits!)
- replies to individual users
- reactions/emojis
- bridging to IRC
- no line limits
- images

The room links in the *General channels* or in the *Working groups* list will take you directly to the relevant rooms.

If there is no appropriate room for your community, please create it.

For more information, see the community-hosted Matrix FAQ.

You can add Matrix shields to your repository's `README.md` using the shield in the community-topics repository as a template.

Ansible community on IRC

The Ansible community maintains several IRC channels on irc.libera.chat. To join the community using IRC, you need one thing:

- an IRC client

IRC chat supports:

- no persistence (you only see messages when you are logged on unless you add a bouncer)
- simple text interface
- bridging from Matrix

Our IRC channels may require you to register your IRC nickname. If you receive an error when you connect or when posting a message, see libera.chat's Nickname Registration guide for instructions. To find all `ansible` specific channels on the libera.chat network, use the following command in your IRC client:

```
/msg alis LIST #ansible* -min 5
```

as described in the libera.chat docs.

Our channels record history on the Matrix side. The channel history can be viewed in a browser - all channels will report an appropriate link to `chat.ansible.im` in their Chanserv entrymsg upon joining the room. Alternatively, a URL of the form `https://chat.ansible.im/#/room/# {IRC channel name}:libera.chat` will also work, for example - for the #ansible-docs channel it would be *https://app.element.io/#/room/#ansible-docs:libera.chat*.

General channels

The clickable links will take you directly to the relevant Matrix room in your browser; room/channel information is also given for use in other clients:

- Community social room and posting news for the Bullhorn newsletter - `Matrix: #social:ansible.com | IRC: #ansible-social`
- General usage and support questions - `Matrix: #users:ansible.com | IRC: #ansible`
- Discussions on developer topics and code related to features or bugs - `Matrix: #devel:ansible.com | IRC: #ansible-devel`
- Discussions on community and collections related topics - `Matrix: #community:ansible.com | IRC: #ansible-community`
- **For public community meetings - `Matrix: #meeting:ansible.im | IRC: #ansible-meeting`**
  - We will generally announce these on one or more of the above mailing lists. See the meeting schedule and agenda page

Working groups

Many of our community Working Groups meet in chat. If you want to get involved in a working group, join the Matrix room or IRC channel where it meets or comment on the agenda.

- AAP Configuration as Code - Matrix: #aap_config_as_code:ansible.com
- Amazon (AWS) Working Group - Matrix: #aws:ansible.com | IRC: `#ansible-aws`
- AWX Working Group - Matrix: #awx:ansible.com | IRC: `#ansible-awx`
- Azure Working Group - Matrix: #azure:ansible.com | IRC: `#ansible-azure`
- Community Working Group (including Meetups) - Matrix: #community:ansible.com | IRC: `#ansible-community`
- Container Working Group - Matrix: #container:ansible.com | IRC: `#ansible-container`
- Contributor Experience Working Group - Matrix: #community:ansible.com | IRC: `#ansible-community`
- DigitalOcean Working Group - Matrix: #digitalocean:ansible.im | IRC: `#ansible-digitalocean`
- Diversity Working Group - Matrix: #diversity:ansible.com | IRC: `#ansible-diversity`
- Docker Working Group - Matrix: #devel:ansible.com | IRC: `#ansible-devel`
- Documentation Working Group - Matrix: #docs:ansible.com | IRC: `#ansible-docs`
- Galaxy Working Group - Matrix: #galaxy:ansible.com | IRC: `#ansible-galaxy`
- JBoss Working Group - Matrix: #jboss:ansible.com | IRC: `#ansible-jboss`
- Kubernetes Working Group - Matrix: #kubernetes:ansible.com | IRC: `#ansible-kubernetes`
- Linode Working Group - Matrix: #linode:ansible.com | IRC: `#ansible-linode`
- Molecule Working Group (testing platform for Ansible playbooks and roles) - Matrix: #molecule:ansible.im | IRC: `#ansible-molecule`
- MySQL Working Group - Matrix: #mysql:ansible.com
- Network Working Group - Matrix: #network:ansible.com | IRC: `#ansible-network`
- PostgreSQL Working Group - Matrix: #postgresql:ansible.com

- Remote Management Working Group - Matrix: #devel:ansible.com | IRC: `#ansible-devel`
- Security Automation Working Group - Matrix: #security-automation:ansible.com | IRC: `#ansible-security`
- Storage Working Group - Matrix: #storage:ansible.com | IRC: `#ansible-storage`
- VMware Working Group - Matrix: #vmware:ansible.com | IRC: `#ansible-vmware`
- Windows Working Group - Matrix: #windows:ansible.com | IRC: `#ansible-windows`
- Ansible developer tools Group - Matrix: #devtools:ansible.com | IRC: `#ansible-devtools`

Want to form a new Working Group?

Regional and Language-specific channels

- Comunidad Ansible en español - Matrix: #espanol:ansible.im | IRC: `#ansible-es`
- Communauté française d'Ansible - Matrix: #francais:ansible.im | IRC: `#ansible-fr`
- Communauté suisse d'Ansible - Matrix: #suisse:ansible.im | IRC: `#ansible-zh`
- European Ansible Community - Matrix: #europe:ansible.im | IRC: `#ansible-eu`

Meetings on chat

The Ansible community holds regular meetings on various topics on Matrix/IRC, and anyone who is interested is invited to participate. For more information about Ansible meetings, consult the meeting schedule and agenda page.

Ansible Community Topics

The Ansible Community Steering Committee uses the community-topics repository to asynchronously discuss with the Community and vote on Community topics in corresponding issues.

Create a new issue in the repository if you want to discuss an idea that impacts any of the following:

- Ansible Community
- Community collection best practices and requirements
- Community collection inclusion policy
- The Community governance
- Other proposals of importance that need the Committee or overall Ansible community attention

Ansible Automation Platform support questions

Red Hat Ansible Automation Platform is a subscription that contains support, certified content, and tooling for Ansible including content management, a controller, UI and REST API.

If you have a question about Ansible Automation Platform, visit Red Hat support rather than using a chat channel or the general project mailing list.

The Bullhorn

The Bullhorn is our newsletter for the Ansible contributor community. Please subscribe to receive it.

If you have any content you would like to share, please contribute/suggest it for upcoming releases.

If you have any questions, please reach out to us at `the-bullhorn@redhat.com`.

Read past issues on the official Bullhorn's wiki page.

How can I help?

Thanks for being interested in helping the Ansible project!

There are many ways to help the Ansible project…but first, please read and understand the *Community Code of Conduct*.

Become a power user

A great way to help the Ansible project is to become a power user:

- Use Ansible everywhere you can
- Take tutorials and classes
- Read the *official documentation*
- Study some of the many excellent books about Ansible
- Get certified.

When you become a power user, your ability and opportunities to help the Ansible project in other ways will multiply quickly.

Ask and answer questions online

There are many forums online where Ansible users ask and answer questions. Reach out and communicate with your fellow Ansible users.

You can find the official *Ansible communication channels*.

Review, fix, and maintain the documentation

Typos are everywhere, even in the Ansible documentation. We work hard to keep the documentation up-to-date, but you may also find outdated examples. We offer easy ways to *report and/or fix documentation errors*.

Participate in your local meetup

There are Ansible meetups all over the world. Join your local meetup. Attend regularly. Ask good questions. Volunteer to give a presentation about how you use Ansible.

If there is no meetup near you, we are happy to help you start one.

File and verify issues

All software has bugs, and Ansible is no exception. When you find a bug, you can help tremendously by telling us about it:

- Filing *issues for ansible-core*.
- Filing *issues for collections*.

If the bug you found already exists in an issue, you can help by verifying the behavior of the reported bug with a comment in that issue, or by reporting any additional information.

Review and submit pull requests

As you become more familiar with how Ansible works, you may be able to fix issues or develop new features yourself. If you think you have a fix for a bug in Ansible, or if you have a new feature that you would like to share with millions of Ansible users, read all about the *development process* to learn how to get your code accepted into Ansible.

You can also get started with solving GitHub issues labeled with the `easyfix` and `good_first_issue` labels for:

- Ansible collections
- All other Ansible projects

When you choose an issue to work on, add a comment directly on the GitHub issue to say you are looking at it and let others know to avoid conflicting work. You can also ask for help in a comment if you need it.

Another good way to help is to review pull requests that other Ansible users have submitted. Ansible core keeps a full list of open pull requests by file, so if a particular module or plugin interests you, you can easily keep track of all the relevant new pull requests and provide testing or feedback. Alternatively, you can review the pull requests for any collections that interest you. Click *Issue tracker* on the collection documentation page to find the issues and PRs for that collection.

Become a collection maintainer

Once you have learned about the development process and have contributed code to a collection, we encourage you to become a maintainer of that collection. There are hundreds of modules in dozens of Ansible collections, and the vast majority of them are written and maintained entirely by members of the Ansible community.

> See *collection maintainer guidelines* to learn more about the responsibilities of being an Ansible collection maintainer.

Join a working group

Working groups are a way for Ansible community members to self-organize around particular topics of interest. We have working groups around various topics. To join or create a working group, please read the *Ansible Working Groups*.

Teach Ansible to others

We are working on a standardized Ansible workshop that can provide a good hands-on introduction to Ansible usage and concepts.

Social media

If you like Ansible and just want to spread the good word, feel free to share on your social media platform of choice, and let us know by using `@ansible` or `#ansible`. We'll be looking for you.

Other ways to get involved

Here are some other ways to connect with the Ansible community:

- **Find an Ansible Meetup near me**
  communication
- Learn more about Ansible:
  - Read books.
  - Get certified.
  - Attend events.
  - Review getting started guides.
  - Watch videos - includes Ansible Automates, AnsibleFest & webinar recordings.
- See where new releases are announced

1.12.2 Contributor path

This section describes the contributor's journey from the beginning to becoming a leader who helps shape the future of Ansible. You can use this path as a roadmap for your long-term participation.

Any contribution to the project, even a small one, is very welcome and valuable. Any contribution counts, whether it's feedback on an issue, a pull request, a topic or documentation change, or a coding contribution. When you contribute regularly, your proficiency and judgment in the related area increase and, along with this, the importance of your presence in the project.

Determine your area of interest

First, determine areas that are interesting to you. Consider your current experience and what you'd like to gain. For example, if you use a specific collection, have a look there. See *How can I help?* for more ideas on how to help.

Find the corresponding project

These are multiple community projects in the Ansible ecosystem you could contribute to:

- Ansible Core
- Collections
- AWX
- Galaxy
- ansible-lint
- Molecule

Learn

The required skillset depends on the area of interest and the project you'll be working on. Remember that the best way to learn is by doing.

Specific knowledge for code developers

Code development requires the most technical knowledge. Let's sort out what an Ansible developer should learn.

You should understand at least the *basics* of the following tools:

- Python programming language
- Git
- GitHub collaborative development model through forks and pull requests

You can learn these tools more in-depth when working on your first contributions.

Each Ansible project has its own set of contributor guidelines. Familiarize yourself with these as you prepare your first contributions.

- *Ansible Core development*.
- *Ansible collection development* and the collection-level contributor guidelines in the collection repository.

Making your first contribution

You can find some ideas on how you can contribute in *How can I help?*.

If you are interested in contributing to collections, take a look at *collection contributions* and the collection repository's `README` and `CONTRIBUTING` files. To make your first experience as smooth as possible, read the repository documentation carefully, then ask the repository maintainers for guidance if you have any questions.

Take a look at GitHub issues labeled with the `easyfix` and `good_first_issue` labels for:

- Ansible collections repositories
- All other Ansible projects

Issues labeled with the `docs` label in Ansible collections and other Ansible projects can be also good to start with.

When you choose an issue to work on, add a comment directly on the GitHub issue to say you are looking at it and let others know to avoid conflicting work. You can also ask for help in a comment if you need it.

Continue to contribute

We don't expect everybody to know everything. Start small, think big. When you contribute regularly, your proficiency and judgment in the related area will improve quickly and, along with this, the importance of your presence in the project.

See *Communicating with the Ansible community* for ways to communicate and engage with the Ansible community, including working group meetings, accessing the Bullhorn news bulletin, and upcoming contributor summits.

Teach others

Share your experience with other contributors through *improving documentation*, answering questions from other contributors and users on *Matrix/Libera.Chat IRC*, giving advice on issues and pull requests, and discussing Community Topics.

Become a collection maintainer

If you are a code contributor to a collection, you can get extended permissions in the repository and become a maintainer. A collection maintainer is a contributor trusted by the community who makes significant and regular contributions to the project and showed themselves as a specialist in the related area. See *Guidelines for collection maintainers* for details.

For some collections that use the collection bot, such as community.general and community.network, you can have different levels of access and permissions.

- *Module maintainers* - The stage prior to becoming a collection maintainer. The file is usually a module or plugin. File maintainers have indirect commit rights.
- supershipit permissions - Similar to being a file maintainer but the scope where a maintainer has the indirect commit is the whole repository.
- `triage` - Access to the repository that allows contributors to manage issues and pull requests.
- `write` access to the repository also known as `commit`. In other words, become a committer. This access level allows contributors to merge pull requests to the development branch as well as perform all the other activities listed in the *Guidelines for collection maintainers*.

For information about permission levels, see the GitHub official documentation.

Become a steering committee member

Note: You do NOT have to be a programmer to become a steering committee member.

The *Steering Committee* member status reflects the highest level of trust which allows contributors to lead the project by making very important decisions for the Ansible project. The Committee members are the community leaders who shape the project's future and the future of automation in the IT world in general.

To reach the status, as the current Committee members did before getting it, along with the things mentioned in this document, you should:

- Subscribe to, comment on, and vote on the Community Topics.
- Propose your topics.
- If time permits, join the Community meetings. Note this is **NOT** a requirement.

1.13 Ansible Collections Contributor Guide

1.13.1 The Ansible Collections Development Cycle

Ansible developers (including community contributors) add new features, fix bugs, and update code in many different repositories. These repositories contain plugins and modules that enable Ansible to execute specific tasks, like adding a user to a particular database or configuring a particular network device. These repositories contain the source code for collections.

Development on collections occurs at the macro and micro levels. Each collection has its own macro development cycle. For more information on the collections development cycle, see *Contributing to Ansible-maintained Collections*. The micro-level lifecycle of a PR is similar in collections and in `ansible-core`.

Macro development: roadmaps, releases, and projects

If you want to follow the conversation about what features will be added to the Ansible package for upcoming releases and what bugs are being fixed, you can watch these resources:

- the *Ansible Roadmap*
- the *Ansible Release Schedule*
- the Ansible Community Working Group .

Micro development: the lifecycle of a PR

If you want to contribute a feature or fix a bug in a collection, you must open a **pull request** ("PR" for short). GitHub provides a great overview of how the pull request process works in general. The ultimate goal of any pull request is to get merged and become part of a collection. Each collection has its own contributor guidelines so please check there for specific details.

Here's an overview of the PR lifecycle:

- *Contributor opens a PR*
- CI runs the test suite
- Developers, maintainers, community review the PR
- Contributor addresses any feedback from reviewers
- Developers, maintainers, community re-review
- PR merged or closed

Making your PR merge-worthy

We do not merge every PR. See *Creating your first collection pull request* for tips to make your PR useful, attractive, and merge-worthy.

Creating changelog fragments

Most changelogs should emphasize the impact of the change on the end user of the feature or collection, unless the change impacts developers directly. Consider what the user needs to know about this change and write the changelog to convey that detail.

Changelogs help users and developers keep up with changes to Ansible collections. Many collections build changelogs for each release from fragments. For collections that use this model, you **must** add a changelog fragment to any PR that changes functionality or fixes a bug.

You do not need a changelog fragment for PRs that:

- add new modules and plugins, because Ansible tooling does that automatically;
- contain only documentation changes.

Note: Some collections require a changelog fragment for every pull request. They use the `trivial:` section for entries mentioned above that will be skipped when building a release changelog.

More precisely:

- Every bugfix PR must have a changelog fragment. The only exception are fixes to a change that has not yet been included in a release.
- Every feature PR must have a changelog fragment.
- New modules and plugins (including jinja2 filter and test plugins) must have `version_added` entries set correctly in their documentation, and do not need a changelog fragment. The tooling detects new modules and plugins by their `version_added` values and announces them in the next release's changelog automatically.

We build short summary changelogs for minor releases as well as for major releases. If you backport a bugfix, include a changelog fragment with the backport PR.

Creating a changelog fragment

A basic changelog fragment is a `.yaml` or `.yml` file placed in the `changelogs/fragments/` directory. Each file contains a yaml dict with keys like `bugfixes` or `major_changes` followed by a list of changelog entries of bugfixes or features. Each changelog entry is rst embedded inside of the yaml file which means that certain constructs would need to be escaped so they can be interpreted by rst and not by yaml (or escaped for both yaml and rst if you prefer). Each PR **must** use a new fragment file rather than adding to an existing one, so we can trace the change back to the PR that introduced it.

PRs which add a new module or plugin do not necessarily need a changelog fragment. See *Creating changelog fragments*. Also see *Changelog fragment entry format* for the precise format changelog fragments should have.

To create a changelog entry, create a new file with a unique name in the `changelogs/fragments/` directory of the corresponding repository. The file name should include the PR number and a description of the change. It must end with the file extension `.yaml` or `.yml`. For example: `40696-user-backup-shadow-file.yaml`

A single changelog fragment may contain multiple sections but most will only contain one section. The toplevel keys (bugfixes, major_changes, and so on) are defined in the config file for our release note tool. Here are the valid sections and a description of each:

breaking_changes

MUST include changes that break existing playbooks or roles. This includes any change to existing behavior that forces users to update tasks. Breaking changes means the user MUST make a change when they update. Breaking changes MUST only happen in a major release of the collection. Write in present tense and clearly describe the new behavior that the end user must now follow. Displayed in both the changelogs and the *Porting Guides*.

```
breaking_changes:
  - ec2_instance - instance wait for state behavior no longer waits for the␣
↪instance monitoring status to become OK when launching a new instance. If plays␣
↪require the old behavior, the action will need to specify ``state: started``␣
↪(https://github.com/ansible-collections/amazon.aws/pull/481).
```

major_changes

Major changes to ansible-core or a collection. SHOULD NOT include individual module or plugin changes. MUST include non-breaking changes that impact all or most of a collection (for example, updates to support a new SDK version across the collection). Major changes mean the user can CHOOSE to make a change when they update but do not have to. Could be used to announce an important upcoming EOL or breaking change in a future release. (ideally 6 months in advance, if known. See this example). Write in present tense and describe what is new. Optionally, include a 'Previously..." sentence to help the user identify where old behavior should now change. Displayed in both the changelogs and the *Porting Guides*.

```
major_changes:
  - bitbucket_* modules - client_id is no longer marked as ``no_log=true``. If you␣
↪relied on its value not showing up in logs and output, mark the whole tasks with␣
↪``no_log: true`` (https://github.com/ansible-collections/community.general/pull/
↪2045).
```

minor_changes

Minor changes to ansible-core, modules, or plugins. This includes new parameters added to modules, or non-breaking behavior changes to existing parameters, such as adding new values to choices[]. Minor changes are enhancements, not bug fixes. Write in present tense.

```
minor_changes:
  - nmcli - adds ``routes6`` and ``route_metric6`` parameters for supporting IPv6␣
↪routes (https://github.com/ansible-collections/community.general/issues/4059).
```

deprecated_features

Features that have been deprecated and are scheduled for removal in a future release. Write in past tense. Include an alternative, where available, for the feature being deprecated. Displayed in both the changelogs and the *Porting Guides*.

```
deprecated_features:
  - mail callback plugin - not specifying ``sender`` is deprecated and will be␣
↪disallowed in ``community.general`` 6.0.0 (https://github.com/ansible-collections/
↪community.general/pull/4140).
```

removed_features

Features that were previously deprecated and are now removed. Write in past tense. Include an alternative, where available, for the feature being deprecated. Displayed in both the changelogs and the *Porting Guides*.

```
removed_features:
  - acme_account_facts - the deprecated redirect has been removed. Use ``community.
↪crypto.acme_account_info`` instead (https://github.com/ansible-collections/
↪community.crypto/pull/290).
```

security_fixes
Fixes that address CVEs or resolve security concerns. MUST use security_fixes for any CVEs. Write in present tense. Include links to CVE information.

```
security_fixes:
  - win_psexec - ensure password is masked in ``psexec_``command return result␣
↪(https://github.com/ansible-collections/community.windows/issues/43).
```

bugfixes
Fixes that resolve issues. SHOULD NOT be used for minor enhancements (use `minor_change` instead). Write in past tense to describe the problem and present tense to describe the fix.

```
bugfixes:
  - apt_repository - fix crash caused by  a timeout. The ``cache.update()`` was␣
↪raising an ``IOError`` because of a timeout in ``apt update`` (https://github.com/
↪ansible/ansible/issues/51995).
```

known_issues
Known issues that are currently not fixed or will not be fixed. Write in present tense to describe the problem and in imperative tense to describe any available workaround.

```
known_issues:
  - idrac_user - module may error out with the message ``unable to perform the␣
↪import or export operation`` because there are pending attribute changes or a␣
↪configuration job is in progress. Wait for the job to complete and run the task␣
↪again.(https://github.com/dell/dellemc-openmanage-ansible-modules/pull/303).
```

trivial
Changes where a formal release changelog entry isn't required. `trivial` changelog fragments are excluded from the published changelog output and may be used for changes such as housekeeping, documentation and test only changes. You can use `trivial` for collections that require a changelog fragment for each pull request.

```
trivial:
  - aws_ec2 - fix broken integration test (https://github.com/ansible-collections/
↪amazon.aws/pull/1269).
```

Each changelog entry must contain a link to its issue between parentheses at the end. If there is no corresponding issue, the entry must contain a link to the PR itself.

Most changelog entries are `bugfixes` or `minor_changes`.

Changelog fragment entry format

When writing a changelog entry, use the following format:

```
- scope - description starting with a lowercase letter and ending with a period at the␣
↪very end. Multiple sentences are allowed (https://github.com/reference/to/an/issue or,␣
↪if there is no issue, reference to a pull request itself).
```

The scope is usually a module or plugin name or group of modules or plugins, for example, `lookup plugins`. While module names can (and should) be mentioned directly (`foo_module`), plugin names should always be followed by the type (`foo inventory plugin`).

For changes that are not really scoped (for example, which affect a whole collection), use the following format:

```
- Description starting with an uppercase letter and ending with a dot at the very end.␣
↪Multiple sentences are allowed (https://github.com/reference/to/an/issue or, if there␣
↪is no issue, reference to a pull request itself).
```

Here are some examples:

```
bugfixes:
  - apt_repository - fix crash caused by ``cache.update()`` raising an ``IOError``
    due to a timeout in ``apt update`` (https://github.com/ansible/ansible/issues/51995).
```

```
minor_changes:
  - lineinfile - add warning when using an empty regexp (https://github.com/ansible/
↪ansible/issues/29443).
```

```
bugfixes:
  - copy - the module was attempting to change the mode of files for
    remote_src=True even if mode was not set as a parameter.  This failed on
    filesystems which do not have permission bits (https://github.com/ansible/ansible/
↪issues/29444).
```

You can find more example changelog fragments in the changelog directory for the community.general development branch.

After you have written the changelog fragment for your PR, commit the file and include it with the pull request.

Changelog fragment entry format for new jinja2 plugins, roles, and playbooks

While new modules and plugins that are not jinja2 filter or test plugins are mentioned automatically in the generated changelog, jinja2 filter and test plugins, roles, and playbooks are not. To make sure they are mentioned, a changelog fragment in a specific format is needed:

```
# A new jinja2 filter plugin:
add plugin.filter:
  - # The following needs to be the name of the filter itself, not of the file
    # the filter is included in!
    name: to_time_unit
    # The description should be in the same format as short_description for
    # other plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
```

(continues on next page)

(continued from previous page)

```
    description: Converts a time expression to a given unit

# A new jinja2 test plugin:
add plugin.test:
  - # The following needs to be the name of the test itself, not of the file
    # the test is included in!
    name: asn1time
    # The description should be in the same format as short_description for
    # other plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: Check whether the given string is an ASN.1 time

# A new role:
add object.role:
  - # This should be the short (non-FQCN) name of the role.
    name: nginx
    # The description should be in the same format as short_description for
    # plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: A nginx installation role

# A new playbook:
add object.playbook:
  - # This should be the short (non-FQCN) name of the playbook.
    name: wipe_server
    # The description should be in the same format as short_description for
    # plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: Wipes a server
```

1.13.2 Requesting changes to a collection

Reporting a bug

Security bugs

Ansible practices responsible disclosure - if this is a security-related bug, email security@ansible.com instead of filing a ticket or posting to any public groups, and you will receive a prompt response.

Bugs in collections

Many bugs only affect a single module or plugin. If you find a bug that affects a module or plugin hosted in a collection, file the bug in the repository of the *collection*:

1. Find the collection on Galaxy.
2. Click on the Issue Tracker link for that collection.
3. Follow the contributor guidelines or instructions in the collection repo.

If you are not sure whether a bug is in ansible-core or in a collection, you can report the behavior on the *mailing list or community chat channel first*.

Requesting a feature

Before you request a feature, check what is *planned for future Ansible Releases*. The best way to get a feature into an Ansible collection is to *submit a pull request*, either against ansible-core or against a collection. See also the *Requirements to merge your PR*.

You can also submit a feature request by opening an issue in the collection repository.

1.13.3 Creating your first collection pull request

This section describes all steps needed to create your first patch and submit a pull request on a collection.

Prepare your environment

Note: These steps assume a Linux work environment with `git` installed.

1. Install and start `docker` or `podman`. This ensures tests run properly isolated and in the same environment as in CI.
2. *Install Ansible or ansible-core*. You need the `ansible-test` utility which is provided by either of these packages.
3. Create the following directories in your home directory:

   ```
   $ mkdir -p ~/ansible_collections/NAMESPACE/COLLECTION_NAME
   ```

   For example, if the collection is `community.mysql`, it would be:

   ```
   $ mkdir -p ~/ansible_collections/community/mysql
   ```

4. Fork the collection repository through the GitHub web interface.
5. Clone the forked repository from your profile to the created path:

   ```
   $ git clone https://github.com/YOURACC/COLLECTION_REPO.git ~/ansible_
   ↪collections/NAMESPACE/COLLECTION_NAME
   ```

   If you prefer to use the SSH protocol:

```
$ git clone git@github.com:YOURACC/COLLECTION_REPO.git ~/ansible_collections/
↪NAMESPACE/COLLECTION_NAME
```

6. Go to your new cloned repository.

```
$ cd ~/ansible_collections/NAMESPACE/COLLECTION_NAME
```

7. Ensure you are in the default branch (it is usually `main`):

```
$ git status
```

8. Show remotes. There should be the `origin` repository only:

```
$ git remote -v
```

9. Add the `upstream` repository:

```
$ git remote add upstream https://github.com/ansible-collections/COLLECTION_
↪REPO.git
```

   This is the repository where you forked from.

10. Update your local default branch. Assuming that it is `main`:

```
$ git fetch upstream
$ git rebase upstream/main
```

11. Create a branch for your changes:

```
$ git checkout -b name_of_my_branch
```

Change the code

Note: Do NOT mix several bug fixes or features that are not tightly related in one pull request. Use separate pull requests for different changes.

You should start with writing integration and unit tests if applicable. These can verify the bug exists (prior to your code fix) and verify your code fixed that bug once the tests pass.

Note: If there are any difficulties with writing or running the tests or you are not sure if the case can be covered, you can skip this step. Other contributors can help you with tests later if needed.

Note: Some collections do not have integration tests. In this case, unit tests are required.

All integration tests are stored in `tests/integration/targets` subdirectories. Go to the subdirectory containing the name of the module you are going to change. For example, if you are fixing the `mysql_user` module in the `community.mysql` collection, its tests are in `tests/integration/targets/test_mysql_user/tasks`.

The `main.yml` file holds test tasks and includes other test files. Look for a suitable test file to integrate your tests or create and include a dedicated test file. You can use one of the existing test files as a draft.

When fixing a bug, write a task that reproduces the bug from the issue.

Put the reported case in the tests, then run integration tests with the following command:

```
$ ansible-test integration name_of_test_subdirectory --docker -v
```

For example, if the test files you changed are stored in `tests/integration/targets/test_mysql_user/`, the command is as follows:

```
$ ansible-test integration test_mysql_user --docker -v
```

You can use the `-vv` or `-vvv` argument if you need more detailed output.

In the examples above, the default test image is automatically downloaded and used to create and run a test container. Use the default test image for platform-independent integration tests such as those for cloud modules.

If you need to run the tests against a specific distribution, see the list of supported container images. For example:

```
$ ansible-test integration name_of_test_subdirectory --docker fedora35 -v
```

Note: If you are not sure whether you should use the default image for testing or a specific one, skip the entire step - the community can help you later. You can also try to use the collection repository's CI to figure out which containers are used.

If the tests ran successfully, there are usually two possible outcomes:

- If the bug has not appeared and the tests have passed successfully, ask the reporter to provide more details. It may not be a bug or can relate to a particular software version used or specifics of the reporter's local environment configuration.
- The bug has appeared and the tests have failed as expected showing the reported symptoms.

Fix the bug

See *Contributing your module to an existing Ansible collection* for some general guidelines about Ansible module development that may help you craft a good code fix for the bug.

Test your changes

1. Install `flake8` (`pip install flake8`, or install the corresponding package on your operating system).
2. Run `flake8` against a changed file:

   ```
   $ flake8 path/to/changed_file.py
   ```

   This shows unused imports, which are not shown by sanity tests, as well as other common issues. Optionally, you can use the `--max-line-length=160` command-line argument.
3. Run sanity tests:

   ```
   $ ansible-test sanity path/to/changed_file.py --docker -v
   ```

   If they failed, look at the output carefully - it is informative and helps to identify a problem line quickly. Sanity failings usually relate to incorrect code and documentation formatting.
4. Run integration tests:

```
$ ansible-test integration name_of_test_subdirectory --docker -v
```

For example, if the test files you changed are stored in `tests/integration/targets/test_mysql_user/`, the command is:

```
$ ansible-test integration test_mysql_user --docker -v
```

You can use the `-vv` or `-vvv` argument if you need more detailed output.

There are two possible outcomes:

- They have failed. Look at the output of the command. Fix the problem in the code and run again. Repeat the cycle until the tests pass.
- They have passed. Remember they failed originally? Our congratulations! You have fixed the bug.

In addition to the integration tests, you can also cover your changes with unit tests. This is often required when integration tests are not applicable to the collection.

We use pytest as a testing framework.

Files with unit tests are stored in the `tests/unit/plugins/` directory. If you want to run unit tests, say, for `tests/unit/plugins/test_myclass.py`, the command is:

```
$ ansible-test units tests/unit/plugins/test_myclass.py --docker
```

If you want to run all unit tests available in the collection, run:

```
$ ansible-test units --docker
```

Submit a pull request

1. Commit your changes with an informative but short commit message:

   ```
   $ git add /path/to/changed/file
   $ git commit -m "module_name_you_fixed: fix crash when ..."
   ```

2. Push the branch to `origin` (your fork):

   ```
   $ git push origin name_of_my_branch
   ```

3. In a browser, navigate to the `upstream` repository (http://github.com/ansible-collections/COLLECTION_REPO).
4. Click the *Pull requests* tab.

   GitHub is tracking your fork, so it should see the new branch in it and automatically offer to create a pull request. Sometimes GitHub does not do it, and you should click the *New pull request* button yourself. Then choose *compare across forks* under the *Compare changes* title.

5. Choose your repository and the new branch you pushed in the right drop-down list. Confirm.

a. Fill out the pull request template with all information you want to mention.

b. Put `Fixes + link to the issue` in the pull request's description.

c. Put `[WIP] + short description` in the pull request's title. Mention the name of the module/plugin you are modifying at the beginning of the description.

d. Click *Create pull request*.

6. Add a *changelog fragment* to the `changelogs/fragments` directory. It will be published in release notes, so users will know about the fix.

   a. Run the sanity test for the fragment:

   ```
   $ansible-test sanity changelogs/fragments/ --docker -v
   ```

   b. If the tests passed, commit and push the changes:

   ```
   $ git add changelogs/fragments/myfragment.yml
   $ git commit -m "Add changelog fragment"
   $ git push origin name_of_my_branch
   ```

7. Verify the CI tests pass that run automatically on Red Hat infrastructure after every commit.

   You will see the CI status at the bottom of your pull request. If they are green and you think that you do not want to add more commits before someone should take a closer look at it, remove `[WIP]` from the title. Mention the issue reporter in a comment and let contributors know that the pull request is "Ready for review".

8. Wait for reviews. You can also ask for a review in the `#ansible-community` *Matrix/Libera.Chat IRC channel*.

9. If the pull request looks good to the community, committers will merge it.

For more in-depth details on this process, see the *Ansible developer guide*.

1.13.4 Testing Collection Contributions

This section focuses on the different tests a contributor should run on their collection PR.

How to test a collection PR

Reviewers and issue authors can verify a PR fixes the reported bug by testing the PR locally.

Prepare your environment

We assume that you use Linux as a work environment (you can use a virtual machine as well) and have `git` installed.

1. *Install Ansible* or ansible-core.

2. Create the following directories in your home directory:

   ```
   mkdir -p ~/ansible_collections/NAMESPACE/COLLECTION_NAME
   ```

   For example, if the collection is `community.general`:

   ```
   mkdir -p ~/ansible_collections/community/general
   ```

   If the collection is `ansible.posix`:

```
mkdir -p ~/ansible_collections/ansible/posix
```

3. Clone the forked repository from the author profile to the created path:

```
git clone https://github.com/AUTHOR_ACC/COLLECTION_REPO.git ~/ansible_
↪collections/NAMESPACE/COLLECTION_NAME
```

4. Go to the cloned repository.

```
cd ~/ansible_collections/NAMESPACE/COLLECTION_NAME
```

5. Checkout the PR branch (it can be retrieved from the PR's page):

```
git checkout pr_branch
```

Test the Pull Request

1. Include *~/ansible_collections* in *COLLECTIONS_PATHS*. See *COLLECTIONS_PATHS* for details.
2. Run your playbook using the PR branch and verify the PR fixed the bug.
3. Give feedback on the pull request or the linked issue(s).

Add unit tests to a collection

This section describes all of the steps needed to add unit tests to a collection and how to run them locally using the `ansible-test` command.

See testing_units_modules for more details.

Understanding the purpose of unit tests

Unit tests ensure that a section of code (known as a `unit`) meets its design requirements and behaves as intended. Some collections do not have unit tests but it does not mean they are not needed.

A `unit` is a function or method of a class used in a module or plugin. Unit tests verify that a function with a certain input returns the expected output.

Unit tests should also verify when a function raises or handles exceptions.

Ansible uses pytest as a testing framework.

See testing_units_modules for complete details.

Inclusion in the Ansible package *requires integration and/or unit tests* You should have tests for your collection as well as for individual modules and plugins to make your code more reliable To learn how to get started with integration tests, see *Adding integration tests to a collection*.

See *Prepare your environment* to prepare your environment.

Determine if unit tests exist

Ansible collection unit tests are located in the `tests/units` directory.

The structure of the unit tests matches the structure of the code base, so the tests can reside in the `tests/units/plugins/modules/` and `tests/units/plugins/module_utils` directories. There can be sub-directories, if modules are organized by module groups.

If you are adding unit tests for `my_module` for example, check to see if the tests already exist in the collection source tree with the path `tests/units/plugins/modules/test_my_module.py`.

Example of unit tests

Let's assume that the following function is in `my_module` :

```
def convert_to_supported(val):
    """Convert unsupported types to appropriate."""
    if isinstance(val, decimal.Decimal):
        return float(val)

    if isinstance(val, datetime.timedelta):
        return str(val)

    if val == 42:
        raise ValueError("This number is just too cool for us ;)")

    return val
```

Unit tests for this function should, at a minimum, check the following:

- If the function gets a `Decimal` argument, it returns a corresponding `float` value.
- If the function gets a `timedelta` argument, it returns a corresponding `str` value.
- If the function gets `42` as an argument, it raises a `ValueError`.
- If the function gets an argument of any other type, it does nothing and returns the same value.

To write these unit tests in collection is called `community.mycollection`:

1. If you already have your local environment *prepared*, go to the collection root directory.

```
cd ~/ansible_collection/community/mycollection
```

2. Create a test file for `my_module`. If the path does not exist, create it.

```
touch tests/units/plugins/modules/test_my_module.py
```

3. Add the following code to the file:

```
# -*- coding: utf-8 -*-

from __future__ import (absolute_import, division, print_function)
__metaclass__ = type
```

(continues on next page)

(continued from previous page)

```
from datetime import timedelta
from decimal import Decimal

import pytest

from ansible_collections.community.mycollection.plugins.modules.my_module␣
↪import (
    convert_to_supported,
)

# We use the @pytest.mark.parametrize decorator to parametrize the function
# https://docs.pytest.org/en/latest/how-to/parametrize.html
# Simply put, the first element of each tuple will be passed to
# the test_convert_to_supported function as the test_input argument
# and the second element of each tuple will be passed as
# the expected argument.
# In the function's body, we use the assert statement to check
# if the convert_to_supported function given the test_input,
# returns what we expect.
@pytest.mark.parametrize('test_input, expected', [
    (timedelta(0, 43200), '12:00:00'),
    (Decimal('1.01'), 1.01),
    ('string', 'string'),
    (None, None),
    (1, 1),
])
def test_convert_to_supported(test_input, expected):
    assert convert_to_supported(test_input) == expected

def test_convert_to_supported_exception():
    with pytest.raises(ValueError, match=r"too cool"):
        convert_to_supported(42)
```

See testing_units_modules for examples on how to mock `AnsibleModule` objects, monkeypatch methods (`module.fail_json`, `module.exit_json`), emulate API responses, and more.

4. Run the tests using docker:

```
ansible-test units tests/unit/plugins/modules/test_my_module.py --docker
```

Recommendations on coverage

Use the following tips to organize your code and test coverage:

- Make your functions simple. Small functions that do one thing with no or minimal side effects are easier to test.
- Test all possible behaviors of a function including exception related ones such as raising, catching and handling exceptions.
- When a function invokes the `module.fail_json` method, passed messages should also be checked.

See also:

testing_units_modules
Unit testing Ansible modules

Testing Ansible
Ansible Testing Guide

Adding integration tests to a collection
Integration testing for collections

testing_integration
Integration tests guide

Testing collections
Testing collections

Resource module integration tests
Resource module integration tests

How to test a collection PR
How to test a pull request locally

Adding integration tests to a collection

This section describes the steps to add integration tests to a collection and how to run them locally using the `ansible-test` command.

Understanding integration tests

Note: Some collections do not have integration tests.

Integration tests are functional tests of modules and plugins. With integration tests, we check if a module or plugin satisfies its functional requirements. Simply put, we check that features work as expected and users get the outcome described in the module or plugin documentation.

There are *two kinds of integration tests* used in collections:

- integration tests that use Ansible roles
- integration tests that use `runme.sh`.

This section focuses on integration tests that use Ansible roles.

Integration tests check modules with playbooks that invoke those modules. The tests pass standalone parameters and their combinations, check what the module or plugin reports with the assert module, and the actual state of the system after each task.

Integration test example

Let's say we want to test the `postgresql_user` module invoked with the `name` parameter. We expect that the module will both create a user based on the provided value of the `name` parameter and will report that the system state has changed. We cannot rely on only what the module reports. To be sure that the user has been created, we query our database with another module to see if the user exists.

```
- name: Create PostgreSQL user and store module's output to the result variable
  community.postgresql.postgresql_user:
    name: test_user
  register: result

- name: Check the module returns what we expect
  assert:
    that:
      - result is changed

- name: Check actual system state with another module, in other words, that the user␣
↪exists
  community.postgresql.postgresql_query:
    query: SELECT * FROM pg_authid WHERE rolename = 'test_user'
  register: query_result

- name: We expect it returns one row, check it
  assert:
    that:
      - query_result.rowcount == 1
```

Details about integration tests

The basic entity of an Ansible integration test is a `target`. The target is an *Ansible role* stored in the `tests/integration/targets` directory of the collection repository. The target role contains everything that is needed to test a module.

The names of targets contain the module or plugin name that they test. Target names that start with `setup_` are usually executed as dependencies before module and plugin targets start execution. See *Creating new integration tests* for details.

To run integration tests, we use the `ansible-test` utility that is included in the `ansible-core` and `ansible` packages. See *Running integration tests* for details. After you finish your integration tests, see to *Creating your first collection pull request* to learn how to submit a pull request.

Preparing for integration tests for collections

To prepare for developing integration tests:

1. *Set up your local environment*.
2. Determine if integration tests already exist.

   ```
   ansible-test integration --list-targets
   ```

If a collection already has integration tests, they are stored in `tests/integration/targets/*` subdirectories of the collection repository.

If you use `bash` and the `argcomplete` package is installed with `pip` on your system, you can also get a full target list.

```
ansible-test integration <tab><tab>
```

Alternately, you can check if the `tests/integration/targets` directory contains a corresponding directory with the same name as the module. For example, the tests for the `postgresql_user` module of the `community.postgresql` collection are stored in the `tests/integration/targets/postgresql_user` directory of the collection repository. If there is no corresponding target there, then that module does not have integration tests. In this case, consider adding integration tests for the module. See *Creating new integration tests* for details.

Recommendations on coverage

Bugfixes

Before fixing code, create a test case in an *appropriate test target* that reproduces the bug provided by the issue reporter and described in the `Steps to Reproduce` issue section. *Run* the tests.

If you failed to reproduce the bug, ask the reporter to provide additional information. The issue may be related to environment settings. Sometimes specific environment issues cannot be reproduced in integration tests, in that case, manual testing by issue reporter or other interested users is required.

Refactoring code

When refactoring code, always check that related options are covered in a *corresponding test target*. Do not assume if the test target exists, everything is covered.

Covering modules / new features

When covering a module, cover all its options separately and their meaningful combinations. Every possible use of the module should be tested against:

- Idempotency - Does rerunning a task report no changes?
- Check-mode - Does dry-running a task behave the same as a real run? Does it not make any changes?
- Return values - Does the module return values consistently under different conditions?

Each test action has to be tested at least the following times:

- Perform an action in check-mode if supported. This should indicate a change.
- Check with another module that the changes have `not` been actually made.
- Perform the action for real. This should indicate a change.
- Check with another module that the changes have been actually made.
- Perform the action again in check-mode. This should indicate `no` change.
- Perform the action again for real. This should indicate `no` change.

To check a task:

1. Register the outcome of the task as a variable, for example, `register:  result`. Using the assert module, check:

1. If `- result is changed` or not.

2. Expected return values.

2. If the module changes the system state, check the actual system state using at least one other module. For example, if the module changes a file, we can check that the file has been changed by checking its checksum with the stat module before and after the test tasks.

3. Run the same task with `check_mode: true` if check-mode is supported by the module. Check with other modules that the actual system state has not been changed.

4. Cover cases when the module must fail. Use the `ignore_errors: true` option and check the returned message with the `assert` module.

Example:

```
- name: Task to fail
  abstract_module:
      ...
  register: result

- name: Check the task fails and its error message
  assert:
    that:
      - result is failed
      - result.msg == 'Message we expect'
```

Here is a summary:

- Cover options and their sensible combinations.
- Check returned values.
- Cover check-mode if supported.
- Check a system state using other modules.
- Check when a module must fail and error messages.

Adding to an existing integration test

The test tasks are stored in the `tests/integration/targets/<target_name>/tasks` directory.

The `main.yml` file holds test tasks and includes other test files. Look for a suitable test file to integrate your tests or create and include or import a separate test file. You can use one of the existing test files as a draft.

When fixing a bug

When fixing a bug:

1. *Determine if integration tests for the module exist*. If they do not, see *Creating new integration tests* section.

2. Add a task that reproduces the bug to an appropriate file within the `tests/integration/targets/<target_name>/tasks` directory.

3. *Run the tests*. The newly added task should fail.

4. If they do not fail, re-check if your environment or test task satisfies the conditions described in the `Steps to Reproduce` section of the issue.

5. If you reproduce the bug and tests fail, change the code.

6. *Run the tests* again.
7. If they fail, repeat steps 5-6 until the tests pass.

Here is an example.

Let's say someone reported an issue in the `community.postgresql` collection that when users pass a name containing underscores to the `postgresql_user` module, the module fails.

We cloned the collection repository to the `~/ansible_collections/community/postgresql` directory and *prepared our environment*. From the collection's root directory, we run `ansible-test integration --list-targets` and it shows a target called `postgresql_user`. It means that we already have tests for the module.

We start with reproducing the bug.

First, we look into the `tests/integration/targets/postgresql_user/tasks/main.yml` file. In this particular case, the file imports other files from the `tasks` directory. The `postgresql_user_general.yml` looks like an appropriate one to add our tests.

```
# General tests:
- import_tasks: postgresql_user_general.yml
  when: postgres_version_resp.stdout is version('9.4', '>=')
```

We will add the following code to the file.

```
# https://github.com/ansible-collections/community.postgresql/issues/NUM
- name: Test user name containing underscore
  community.postgresql.postgresql_user:
    name: underscored_user
  register: result

- name: Check the module returns what we expect
  assert:
    that:
      - result is changed

- name: Query the database if the user exists
  community.postgresql.postgresql_query:
    query: SELECT * FROM pg_authid WHERE rolename = 'underscored_user'
  register: result

- name: Check the database returns one row
  assert:
    that:
      - result.query_result.rowcount == 1
```

When we *run the tests* with `postgresql_user` as a test target, this task must fail.

Now that we have our failing test; we will fix the bug and run the same tests again. Once the tests pass, we will consider the bug fixed and will submit a pull request.

When adding a new feature

Note: The process described in this section also applies when you want to add integration tests to a feature that already exists, but is missing integration tests.

If you have not already implemented the new feature, you can start by writing the integration tests for it. They will not work as the code does not yet exist, but they can help you improve your implementation design before you start writing any code.

When adding new features, the process of adding tests consists of the following steps:

1. *Determine if integration tests for the module exists*. If they do not, see *Creating new integration tests*.
2. Find an appropriate file for your tests within the `tests/integration/targets/<target_name>/tasks` directory.
3. Cover your feature with tests. Refer to the *Recommendations on coverage* section for details.
4. *Run the tests*.
5. If they fail, see the test output for details. Fix your code or tests and run the tests again.
6. Repeat steps 4-5 until the tests pass.

Here is an example.

Let's say we decided to add a new option called `add_attribute` to the `postgresql_user` module of the `community.postgresql` collection.

The option is boolean. If set to `yes`, it adds an additional attribute to a database user.

We cloned the collection repository to the `~/ansible_collections/community/postgresql` directory and *prepared our environment*. From the collection's root directory, we run `ansible-test integration --list-targets` and it shows a target called `postgresql_user`. Therefore, we already have some tests for the module.

First, we look at the `tests/integration/targets/<target_name>/tasks/main.yml` file. In this particular case, the file imports other files from the `tasks` directory. The `postgresql_user_general.yml` file looks like an appropriate one to add our tests.

```
# General tests:
- import_tasks: postgresql_user_general.yml
  when: postgres_version_resp.stdout is version('9.4', '>=')
```

We will add the following code to the file.

```
# https://github.com/ansible-collections/community.postgresql/issues/NUM
# We should also run the same tasks with check_mode: true. We omit it here for␣
↪simplicity.
- name: Test for new_option, create new user WITHOUT the attribute
  community.postgresql.postgresql_user:
    name: test_user
  register: result

- name: Check the module returns what we expect
  assert:
    that:
      - result is changed
```

(continues on next page)

(continued from previous page)

```
- name: Query the database if the user exists but does not have the attribute (it is␣
↪NULL)
  community.postgresql.postgresql_query:
    query: SELECT * FROM pg_authid WHERE rolename = 'test_user' AND attribute = NULL
  register: result

- name: Check the database returns one row
  assert:
    that:
      - result.query_result.rowcount == 1

- name: Test for new_option, create new user WITH the attribute
  community.postgresql.postgresql_user:
    name: test_user
  register: result

- name: Check the module returns what we expect
  assert:
    that:
      - result is changed

- name: Query the database if the user has the attribute (it is TRUE)
  community.postgresql.postgresql_query:
    query: SELECT * FROM pg_authid WHERE rolename = 'test_user' AND attribute = 't'
  register: result

- name: Check the database returns one row
  assert:
    that:
      - result.query_result.rowcount == 1
```

Then we *run the tests* with `postgresql_user` passed as a test target.

In reality, we would alternate the tasks above with the same tasks run with the `check_mode: true` option to be sure our option works as expected in check-mode as well. See *Recommendations on coverage* for details.

If we expect a task to fail, we use the `ignore_errors: true` option and check that the task actually failed and returned the message we expect:

```
- name: Test for fail_when_true option
  community.postgresql.postgresql_user:
    name: test_user
    fail_when_true: true
  register: result
  ignore_errors: true

- name: Check the module fails and returns message we expect
  assert:
    that:
      - result is failed
      - result.msg == 'The message we expect'
```

Running integration tests

In the following examples, we will use Docker to run integration tests locally. Ensure you *Prepare your environment* first.

We assume that you are in the ~/ansible_collections/NAMESPACE/COLLECTION directory.

After you change the tests, you can run them with the following command:

```
ansible-test integration <target_name> --docker <distro>
```

The target_name is a test role directory containing the tests. For example, if the test files you changed are stored in the tests/integration/targets/postgresql_info/ directory and you want to use the fedora34 container image, then the command will be:

```
ansible-test integration postgresql_info --docker fedora34
```

You can use the -vv or -vvv argument if you need more detailed output.

In the examples above, the fedora34 test image will be automatically downloaded and used to create and run a test container.

See the list of supported container images.

In some cases, for example, for platform-independent tests, the default test image is required. Use the --docker default or just --docker option without specifying a distribution in this case.

Note: If you have any difficulties with writing or running integration tests or you are not sure if the case can be covered, submit your pull request without the tests. Other contributors can help you with them later if needed.

Creating new integration tests

This section covers the following cases:

- There are no integration tests for a collection or group of modules in a collection at all.
- You are adding a new module and you want to include integration tests.
- You want to add integration tests for a module that already exists without integration tests.

In other words, there are currently no tests for a module regardless of whether the module exists or not.

If the module already has tests, see *Adding to an existing integration test*.

Simplified example

Here is a simplified abstract example.

Let's say we are going to add integration tests to a new module in the community.abstract collection which interacts with some service.

We *checked* and determined that there are no integration tests at all.

We should basically do the following:

1. Install and run the service with a setup target.
2. Create a test target.

3. Add integration tests for the module.
4. *Run the tests*.
5. Fix the code and tests as needed, run the tests again, and repeat the cycle until they pass.

Note: You can reuse the `setup` target when implementing other targets that also use the same service.

1. Clone the collection to the `~/ansible_collections/community.abstract` directory on your local machine.
2. From the `~/ansible_collections/community.abstract` directory, create directories for the `setup` target:

```
mkdir -p tests/integration/targets/setup_abstract_service/tasks
```

3. Write all the tasks needed to prepare the environment, install, and run the service.

For simplicity, let's imagine that the service is available in the native distribution repositories and no sophisticated environment configuration is required.

Add the following tasks to the `tests/integration/targets/setup_abstract_service/tasks/main.yml` file to install and run the service:

```
- name: Install abstract service
  package:
    name: abstract_service

- name: Run the service
  systemd:
    name: abstract_service
    state: started
```

This is a very simplified example.

4. Add the target for the module you are testing.

Let's say the module is called `abstract_service_info`. Create the following directory structure in the target:

```
mkdir -p tests/integration/targets/abstract_service_info/tasks
mkdir -p tests/integration/targets/abstract_service_info/meta
```

Add all of the needed subdirectories. For example, if you are going to use defaults and files, add the `defaults` and `files` directories, and so on. The approach is the same as when you are creating a role.

5. To make the `setup_abstract_service` target run before the module's target, add the following lines to the `tests/integration/targets/abstract_service_info/meta/main.yml` file.

```
dependencies:
  - setup_abstract_service
```

6. Start with writing a single stand-alone task to check that your module can interact with the service.

We assume that the `abstract_service_info` module fetches some information from the `abstract_service` and that it has two connection parameters.

Among other fields, it returns a field called `version` containing a service version.

Add the following to `tests/integration/targets/abstract_service_info/tasks/main.yml`:

```
- name: Fetch info from abstract service
  abstract_service_info:
    host: 127.0.0.1  # We assume the service accepts local connection by default
    port: 1234       # We assume that the service is listening to this port by default
  register: result   # This variable will contain the returned JSON including the server␣
↪version

- name: Test the output
  assert:
    that:
      - result.version == '1.0.0'  # Check version field contains what we expect
```

7. *Run the tests* with the `-vvv` argument.

If there are any issues with connectivity (for example, the service is not accepting connections) or with the code, the play will fail.

Examine the output to see at which step the failure occurred. Investigate the reason, fix it, and run again. Repeat the cycle until the test passes.

8. If the test succeeds, write more tests. Refer to the *Recommendations on coverage* section for details.

`community.postgresql` example

Here is a real example of writing integration tests from scratch for the `community.postgresql.postgresql_info` module.

For the sake of simplicity, we will create very basic tests which we will run using the Ubuntu 20.04 test container.

We use `Linux` as a work environment and have `git` and `docker` installed and running.

We also installed `ansible-core`.

1. Create the following directories in your home directory:

```
mkdir -p ~/ansible_collections/community
```

2. Fork the collection repository through the GitHub web interface.
3. Clone the forked repository from your profile to the created path:

```
git clone https://github.com/YOURACC/community.postgresql.git ~/ansible_collections/
↪community/postgresql
```

If you prefer to use the SSH protocol:

```
git clone git@github.com:YOURACC/community.postgresql.git ~/ansible_collections/
↪community/postgresql
```

4. Go to the cloned repository:

```
cd ~/ansible_collections/community/postgresql
```

5. Be sure you are in the default branch:

```
git status
```

6. Checkout a test branch:

```
git checkout -b postgresql_info_tests
```

7. Since we already have tests for the `postgresql_info` module, we will run the following command:

```
rm -rf tests/integration/targets/*
```

With all of the targets now removed, the current state is as if we do not have any integration tests for the `community.postgresql` collection at all. We can now start writing integration tests from scratch.

8. We will start with creating a `setup` target that will install all required packages and will launch PostgreSQL. Create the following directories:

```
mkdir -p tests/integration/targets/setup_postgresql_db/tasks
```

9. Create the `tests/integration/targets/setup_postgresql_db/tasks/main.yml` file and add the following tasks to it:

```
- name: Install required packages
  package:
    name:
      - apt-utils
      - postgresql
      - postgresql-common
      - python3-psycopg2

- name: Initialize PostgreSQL
  shell: . /usr/share/postgresql-common/maintscripts-functions && set_system_locale && /
↪usr/bin/pg_createcluster -u postgres 12 main
  args:
    creates: /etc/postgresql/12/

- name: Start PostgreSQL service
  ansible.builtin.service:
    name: postgresql
    state: started
```

That is enough for our very basic example.

10. Then, create the following directories for the `postgresql_info` target:

```
mkdir -p tests/integration/targets/postgresql_info/tasks tests/integration/targets/
↪postgresql_info/meta
```

11. To make the `setup_postgresql_db` target run before the `postgresql_info` target as a dependency, create the `tests/integration/targets/postgresql_info/meta/main.yml` file and add the following code to it:

```
dependencies:
  - setup_postgresql_db
```

12. Now we are ready to add our first test task for the `postgresql_info` module. Create the `tests/integration/targets/postgresql_info/tasks/main.yml` file and add the following code to it:

```
- name: Test postgresql_info module
  become: true
  become_user: postgres
  community.postgresql.postgresql_info:
    login_user: postgres
    login_db: postgres
  register: result

- name: Check the module returns what we expect
  assert:
    that:
      - result is not changed
      - result.version.major == 12
      - result.version.minor == 8
```

In the first task, we run the `postgresql_info` module to fetch information from the database we installed and launched with the `setup_postgresql_db` target. We are saving the values returned by the module into the `result` variable.

In the second task, we check the `result` variable, which is what the first task returned, with the `assert` module. We expect that, among other things, the result has the version and reports that the system state has not been changed.

13. Run the tests in the Ubuntu 20.04 docker container:

```
ansible-test integration postgresql_info --docker ubuntu2004 -vvv
```

The tests should pass. If we look at the output, we should see something like the following:

```
TASK [postgresql_info : Check the module returns what we expect] ***************
ok: [testhost] => {
  "changed": false,
  "msg": "All assertions passed"
}
```

If your tests fail when you are working on your project, examine the output to see at which step the failure occurred. Investigate the reason, fix it, and run again. Repeat the cycle until the test passes. If the test succeeds, write more tests. Refer to the *Recommendations on coverage* section for details.

See also:

testing_units_modules
  Unit testing Ansible modules

pytest
  Pytest framework documentation

Testing Ansible
  Ansible Testing Guide

Add unit tests to a collection
  Unit testing for collections

testing_integration
  Integration tests guide

Testing collections
  Testing collections

Resource module integration tests
  Resource module integration tests

How to test a collection PR
How to test a pull request locally

1.13.5 Review checklist for collection PRs

Use this section as a checklist reminder of items to review when you review a collection PR.

Reviewing bug reports

When users report bugs, verify the behavior reported. Remember always to be kind with your feedback.

- Did the user make a mistake in the code they put in the Steps to Reproduce issue section? We often see user errors reported as bugs.
- Did the user assume an unexpected behavior? Ensure that the related documentation is clear. If not, the issue is useful to help us improve documentation.
- Is there a minimal reproducer? If not, ask the reporter to reduce the complexity to help pinpoint the issue.
- Is the issue a consequence of a misconfigured environment?
- If it seems to be a real bug, does the behaviour still exist in the most recent release or the development branch?
- Reproduce the bug, or if you do not have a suitable infrastructure, ask other contributors to reproduce the bug.

Reviewing suggested changes

When reviewing PRs, verify that the suggested changes do not:

- Unnecessarily break backward compatibility.
- Bring more harm than value.
- Introduce non-idempotent solutions.
- Duplicate already existing features (inside or outside the collection).
- Violate the *Ansible development conventions*.

Other standards to check for in a PR include:

- A pull request MUST NOT contain a mix of bug fixes and new features that are not tightly related. If yes, ask the author to split the pull request into separate PRs.
- If the pull request is not a documentation fix, it must include a *changelog fragment*. Check the format carefully as follows:
- New modules and plugins (that are not jinja2 filter and test plugins) do not need changelog fragments.
- For jinja2 filter and test plugins, check out the special syntax for changelog fragments.
- The changelog content contains useful information for end users of the collection.
- If new files are added with the pull request, they follow the *Collection licensing requirements*.
- The changes follow the *Ansible documentation standards* and the *Ansible documentation style guide*.
- The changes follow the *Development conventions*.
- If a new plugin is added, it is one of the *allowed plugin types*.
- Documentation, examples, and return sections use FQCNs for the `M(..)` *format macros* when referring to modules.

- Modules and plugins from ansible-core use `ansible.builtin.` as an FQCN prefix when mentioned.
- When a new option, module, plugin, or return value is added, the corresponding documentation or return sections use `version_added:` containing the *collection* version in which they will be first released.
- This is typically the next minor release, sometimes the next major release. For example: if 2.7.5 is the current release, the next minor release will be 2.8.0, and the next major release will be 3.0.0).
- FQCNs are used for `extends_documentation_fragment:`, unless the author is referring to doc_fragments from ansible-core.
- New features have corresponding examples in the *EXAMPLES block*.
- Return values are documented in the *RETURN block*.

Review tests in the PR

Review the following if tests are applicable and possible to implement for the changes included in the PR:

- Where applicable, the pull request has testing_integration and testing_units.
- All changes are covered. For example, a bug case or a new option separately and in sensible combinations with other options.
- Integration tests cover `check_mode` if supported.
- Integration tests check the actual state of the system, not only what the module reports. For example, if the module actually changes a file, check that the file was changed by using the `ansible.builtin.stat` module..
- Integration tests check return values, if applicable.

Review for merge commits and breaking changes

- The pull request does not contain merge commits. See the GitHub warnings at the bottom of the pull request. If merge commits are present, ask the author to rebase the pull request branch.
- If the pull request contains breaking changes, ask the author and the collection maintainers if it really is needed, and if there is a way not to introduce breaking changes. If breaking changes are present, they MUST only appear in the next major release and MUST NOT appear in a minor or patch release. The only exception is breaking changes caused by security fixes that are absolutely necessary to fix the security issue.

1.13.6 Ansible community package collections requirements

This section describes the requirements for maintainers of Ansible community collections in the ansible-collections repository or included in the Ansible community package.

Overview

This section provides help, advice, and guidance on making sure your collections are correct and ready for inclusion in the Ansible community package.

Note: Inclusion of a new collection in the Ansible package is ultimately at the discretion of the *Ansible Community Steering Committee*. Every rejected candidate will get feedback. Differences of opinion should be taken to a dedicated Community Topic for discussion and a final vote.

Feedback and communications

As with any project it is very important that we get feedback from users, contributors, and maintainers. You can get feedback and help as follows:

- Discussing in the #community:ansible.com Matrix room, which is bridged with the `#ansible-community` channel on Libera.Chat IRC. See the *Ansible Communication Guide* for details.
- Discussing in the Community Working Group meeting.
- Creating GitHub Issues in the `ansible-collections` repository.

Keeping informed

You should subscribe to:

- The news-for-maintainers repository to track changes that collection maintainers should be aware of. Subscribe only to issues if you want less traffic.
- The Bullhorn Ansible contributor newsletter.

Collection infrastructure

The following guidelines describe the required structure for your collection:

- MUST have a publicly available issue tracker that does not require a paid level of service to create an account or view issues.
- MUST have a Code of Conduct (CoC).
  - The collection's CoC MUST be compatible with the *Community Code of Conduct*.
  - The collections SHOULD consider using the Ansible CoC if they do not have a CoC that they consider better.
  - The *Diversity and Inclusion working group* may evaluate all CoCs and object to a collection's inclusion based on the CoCs contents.
  - The CoC MUST be linked from the `README.md` file, or MUST be present or linked from the `CODE_OF_CONDUCT.md` file in the collection root.
- MUST be published to Ansible Galaxy.
- SHOULD NOT contain any large objects (binaries) comparatively to the current Galaxy tarball size limit of 20 MB, For example, do not include package installers for testing purposes.
- SHOULD NOT contain any unnecessary files such as temporary files.
- MUST only contain objects that follow the *Licensing rules*.

Python Compatibility

A collection MUST be developed and tested using the below Python requirements as Ansible supports a wide variety of machines.

The collection should adhere to the tips at ansible-and-python-3.

Python Requirements

Python requirements for a collection vary between **controller environment** and **other environment**. On the controller-environment, the Python versions required may be higher than what is required on the other-environment. While developing a collection, you need to understand the definitions of both the controller-environment and other-environment to help you choose Python versions accordingly:

- controller environment: The plugins/modules always run in the same environment (Python interpreter, venv, host, and so on) as ansible-core itself.
- other environment: It is possible, even if uncommon in practice, for the plugins/modules to run in a different environment than ansible-core itself.

One example scenario where the "even if" clause comes into play is when using cloud modules. These modules mostly run on the controller node but in some environments, the controller might run on one machine inside a demilitarized zone which cannot directly access the cloud machines. The user has to have the cloud modules run on a bastion host/jump server which has access to the cloud machines.

Controller environment

In the controller environment, collections MUST support Python 2 (version 2.7) and Python 3 (Version 3.6 and higher), unless required libraries do not support these versions. Collections SHOULD also support Python v3.5 if all required libraries support this version.

Other environment

In the other environment, collections MUST support Python 2 (version 2.7) and Python 3 (Version 3.6 and higher), unless required libraries do not support these versions. Collections SHOULD also support Python v2.6 and v3.5 if all required libraries support this version.

Note: If the collection does not support Python 2.6 and/or Python 3.5 explicitly then take the below points into consideration:

- Dropping support for Python 2.6 in the other environment means that you are dropping support for RHEL6. RHEL6 ended full support in November, 2020, but some users are still using RHEL6 under extended support contracts (ELS) until 2024. ELS is not full support; not all CVEs of the python-2.6 interpreter are fixed, for instance.
- Dropping support for Python 3.5 means that Python 2.7 has to be installed on Ubuntu Xenial (16.04) and that you have to support Python 2.7.

Also, note that dropping support for a Python version for an existing module/plugin is a breaking change, and thus requires a major release. A collection MUST announce dropping support for Python versions in their changelog, if possible in advance (for example, in previous versions before support is dropped).

Python documentation requirements

- If everything in your collection supports the same Python versions as the collection-supported versions of ansible-core, you do not need to document Python versions.
- If your collection does not support those Python versions, you MUST document which versions it supports in the README.
- If most of your collection supports the same Python versions as ansible-core, but some modules and plugins do not, you MUST include the supported Python versions in the documentation for those modules and plugins.

For example, if your collection supports Ansible 2.9 to ansible-core 2.13, the Python versions supported for modules are 2.6, 2.7, and 3.5 and newer (until at least 3.10), while the Python versions supported for plugins are 2.7 and 3.5 and newer (until at least 3.10). So if the modules in your collection do not support Python 2.6, you have to document this in the README, for example `The content in this collection supports Python 2.7, Python 3.5 and newer.`.

Standards for developing module and plugin utilities

- `module_utils` and `plugin_utils` can be marked for only internal use in the collection, but they MUST document this and MUST use a leading underscore for filenames.
- It is a breaking change when you make an existing `module_utils` private and in that case the collection requires a major version bump.
- Below are some recommendations for `module_utils` documentation:
  - No docstring: everything we recommend for `other-environment` is supported.
  - The docstring `'Python versions supported:  same as for controller-environment'`: everything we recommend for `controller-environment` is supported.
  - The docstring with specific versions otherwise: `'Python versions supported:  '`.

Repository structure requirements

galaxy.yml

- The `tags` field MUST be set.
- Collection dependencies must meet a set of rules. See the section on *Collection Dependencies <collection_dependencies_>* for details.
- The `ansible` package MUST NOT depend on collections not shipped in the package.
- If you plan to split up your collection, the new collection MUST be approved for inclusion before the smaller collections replace the larger in Ansible.
- If you plan to add other collections as dependencies, they MUST run through the formal application process.

README.md

Your collection repository MUST have a `README.md` in the root of the collection, see collection_template/README.md for an example.

meta/runtime.yml

Example: meta/runtime.yml

- The `meta/runtime.yml` MUST define the minimum version of Ansible which this collection works with.
  - If the collection works with Ansible 2.9, then this should be set to *>=2.9.10*
  - It is usually better to avoid adding *<2.11* as a restriction, since this for example makes it impossible to use the collection with the current ansible-base devel branch (which has version 2.11.0.dev0)

Modules & Plugins

- Collections MUST only use the directories specified below in the `plugins/` directory and only for the purposes listed:

  Those recognized by ansible-core
  : `doc_fragments`, `modules`, `module_utils`, `terminal`, and those listed in *Working with plugins*. This list can be verified by looking at the last element of the package argument of each `*_loader` in https://github.com/ansible/ansible/blob/devel/lib/ansible/plugins/loader.py#L1126

  plugin_utils
  : For shared code which is only used controller-side, not in modules.

  sub_plugins
  : For other plugins which are managed by plugins inside of collections instead of ansible-core. We use a subfolder so there aren't conflicts when ansible-core adds new plugin types.

  The core team (which maintains ansible-core) has committed not to use these directories for anything which would conflict with the uses specified here.

Other directories

Collections MUST not use files outside `meta/`, `plugins/`, `roles/` and `playbooks/` in any plugin, role, or playbook that can be called by FQCN, used from other collections, or used from user playbooks and roles. A collection must work if every file or directory is deleted from the installed collection except those four directories and their contents.

Internal plugins, roles and playbooks (artifacts used only in testing, or only to release the collection, or only for some other internal purpose and not used externally) are exempt from this rule and may rely on files in other directories.

Documentation requirements

All modules and plugins MUST:

- Include a DOCUMENTATION block.
- Include an EXAMPLES block (except where not relevant for the plugin type).
- Use FQCNs when referring to modules, plugins and documentation fragments inside and outside the collection (including `ansible.builtin` for the listed entities from ansible-core.

When using `version_added` in the documentation:

- Declare the version of the collection in which the options were added – NOT the version of Ansible.
- If you for some reason really have to specify version numbers of Ansible or of another collection, you also have to provide `version_added_collection: collection_name`. We strongly recommend to NOT do this.
- Include `version_added` when you add new content (modules, plugins, options) to an existing collection. The values are shown in the documentation, and can be useful, but you do not need to add `version_added` to every option, module, and plugin when creating a new collection.

Other items:

- The `CONTRIBUTING.md` (or `README.md`) file MUST state what types of contributions (pull requests, feature requests, and so on) are accepted and any relevant contributor guidance. Issues (bugs and feature request) reports must always be accepted.
- Collections are encouraged to use z:ref:*links and formatting macros <linking-and-other-format-macros-within-module-documentation>*
- Including a RETURN block for modules is strongly encouraged but not required.

Contributor Workflow

Changelogs

Collections are required to include a changelog. To give a consistent feel for changelogs across collections and ensure changelogs exist for collections included in the `ansible` package we suggest you use antsibull-changelog to maintain and generate this but other options exist. Preferred (in descending order):

1. Use antsibull-changelog (preferred).
2. Provide `changelogs/changelog.yaml` in the correct format. (You can use `antsibull-lint changelog-yaml /path/to/changelog.yaml` to validate the format.)
3. Provide a link to the changelog file (self-hosted) (not recommended).

Note that the porting guide is compiled from `changelogs/changelog.yaml` (sections `breaking_changes`, `major_changes`, `deprecated_features`, `removed_features`). So if you use option 3, you will not be able to add something to the porting guide.

Versioning and deprecation

- Collections MUST adhere to semantic versioning.
- To preserve backward compatibility for users, every Ansible minor version series (x.Y.z) will keep the major version of a collection constant. If ansible 3.0.0 includes `community.general` 2.2.0, then each 3.Y.z (3.1.z, 3.2.z, and so on) release will include the latest `community.general` 2.y.z release available at build time. Ansible 3.y.z will **never** include a `community.general` 3.y.z release, even if it is available. Major collection version changes will be included in the next Ansible major release (4.0.0 in this example).
- Therefore, ensure that the current major release of your collection included in 3.0.0 receives at least bugfixes as long as new 3.Y.Z releases are produced.
- Since new minor releases are included, you can include new features, modules and plugins. You must make sure that you do not break backwards compatibility! (See semantic versioning.) This means in particular:
  - You can fix bugs in patch releases, but not add new features or deprecate things.
  - You can add new features and deprecate things in minor releases, but not remove things or change behavior of existing features.
  - You can only remove things or make breaking changes in major releases.
- We recommend that you ensure that if a deprecation is added in a collection version that is included in Ansible 3.y.z, the removal itself will only happen in a collection version included in Ansible 5.0.0 or later, but not in a collection version included in Ansible 4.0.0.
- Content moved from ansible/ansible that was scheduled for removal in 2.11 or later MUST NOT be removed in the current major release available when ansible 2.10.0 is released. Otherwise it would already be removed in 2.10, unexpectedly for users! Deprecation cycles can be shortened (since they are now uncoupled from ansible or ansible-base versions), but existing ones must not be unexpectedly terminated.
- We recommend you announce your policy of releasing, versioning and deprecation to contributors and users in some way. For an example of how to do this, see the announcement in community.general. You could also do this in the README.

Naming

Collection naming

For collections under ansible-collections the repository SHOULD be named `NAMESPACE.COLLECTION`.

To create a new collection and corresponding repository, first, a new namespace in Galaxy has to be created by submitting Request a namespace.

Namespace limitations lists requirements for namespaces in Galaxy.

For collections created for working with a particular entity, they should contain the entity name, for example `community.mysql`.

For corporate maintained collections, the repository can be named `COMPANY_NAME.PRODUCT_NAME`, for example `ibm.db2`.

We should avoid FQCN / repository names:

- which are unnecessary long: try to make it compact but clear.
- contain the same words / collocations in `NAMESPACE` and `COLLECTION` parts, for example `my_system.my_system`.

If your collection is planned to be certified on **Red Hat Automation Hub**, please consult with Red Hat Partner Engineering through `ansiblepartners@redhat.com` to ensure collection naming compatibility between the community collection on **Galaxy**.

Module naming

Modules that only gather information MUST be named `<something>_info`. Modules that return `ansible_facts` are named `<something>_facts` and do not return non-facts. For more information, refer to the Developing modules guidelines.

Collection licensing requirements

Note: The guidelines below are more restrictive than strictly necessary. We will try to add a larger list of acceptable licenses once we have approval from Red Hat Legal.

There are four types of content in collections which licensing has to address in different ways:

modules
must be licensed with a free software license that is compatible with the GPL-3.0-or-later

module_utils
must be licensed with a free software license that is compatible with the GPL-3.0-or-later. Ansible itself typically uses the BSD-2-clause license to make it possible for third-party modules which are licensed incompatibly with the GPLv3 to use them. Please consider this use case when licensing your own `module_utils`.

All other code in `plugins/`
All other code in `plugins/` must be under the GPL-3.0-or-later. These plugins are run inside of the Ansible controller process which is licensed under the `GPL-3.0-or-later` and often must import code from the controller. For these reasons, `GPL-3.0-or-later` must be used.

All other code
Code outside `plugins/` may be licensed under another free software license that is compatible with the GPL-3.0-or-later, provided that such code does not import any other code that is licensed under the `GPL-3.0-or-later`. If the file does import other `GPL-3.0-or-later` code, then it must similarly be licensed under `GPL-3.0-or-later`. Note that this applies in particular to unit tests; these often import code from ansible-core, plugins, module utils, or modules, and such code is often licensed under `GPL-3.0-or-later`.

Non code content
At the moment, these must also be under the GPL-3.0-or-later.

Use this table of licenses from the Fedora Project to find which licenses are compatible with the GPLv3+. The license must be considered open source on both the Fedora License table and the Debian Free Software Guidelines to be allowed.

These guidelines are the policy for inclusion in the Ansible package and are in addition to any licensing and legal concerns that may otherwise affect your code.

Repository management

Every collection MUST have a public git repository. Releases of the collection MUST be tagged in said repository. This means that releases MUST be `git taged` and that the tag name MUST exactly match the Galaxy version number. Tag names MAY have a `v` prefix, but a collection's tag names MUST have a consistent format from release to release.

Additionally, collection artifacts released to Galaxy MUST be built from the sources that are tagged in the collection's git repository as that release. Any changes made during the build process MUST be clearly documented so the collection artifact can be reproduced.

We are open to allowing other SCM software once our tooling supports them.

Branch name and configuration

This subsection is **only** for repositories under ansible-collections! Other collection repositories can also follow these guidelines, but do not have to.

All new repositories MUST have `main` as the default branch.

Existing repositories SHOULD be converted to use `main`.

Repository Protections:

- Allow merge commits: disallowed

Branch protections MUST be enforced:

- Require linear history
- Include administrators

CI Testing

Note: You can copy the free-to-use GitHub action workflow file from the Collection Template repository to the *.github/workflows* directory in your collection to set up testing through GitHub actions. The workflow covers all the requirements below.

- You MUST run the `ansible-test sanity` command from the latest stable ansible-base/ansible-core branch.
  - Collections MUST run an equivalent of the `ansible-test sanity --docker` command.
  - If they do not use `--docker`, they must make sure that all tests run, in particular the compile and import tests (which should run for all supported Python versions).
  - Collections can choose to skip certain Python versions that they explicitly do not support; this needs to be documented in `README.md` and in every module and plugin (hint: use a docs fragment). However we strongly recommend you follow the Ansible Python Compatibility section for more details.
- You SHOULD suggest to *additionally* run `ansible-test sanity` from the ansible/ansible `devel` branch so that you find out about new linting requirements earlier.
- The sanity tests MUST pass.
  - Adding some entries to the `test/sanity/ignore*.txt` file is an allowed method of getting them to pass, except cases listed below.
  - You SHOULD not have ignored test entries. A reviewer can manually evaluate and approve your collection if they deem an ignored entry to be valid.

  - **You MUST not ignore the following validations. They must be fixed before approval:**
    - `validate-modules:doc-choices-do-not-match-spec`
    - `validate-modules:doc-default-does-not-match-spec`
    - `validate-modules:doc-missing-type`
    - `validate-modules:doc-required-mismatch`
    - `validate-modules:mutually_exclusive-unknown`
    - `validate-modules:no-log-needed` (use `no_log=False` in the argument spec to flag false positives!)
    - `validate-modules:nonexistent-parameter-documented`
    - `validate-modules:parameter-list-no-elements`
    - `validate-modules:parameter-type-not-in-doc`
    - `validate-modules:undocumented-parameter`
  - All entries in ignores.txt MUST have a justification in a comment in the ignore.txt file for each entry. For example `plugins/modules/docker_container.py use-argspec-type-path # uses colon-separated paths, can't use type=path`.
  - Reviewers can block acceptance of a new collection if they don't agree with the ignores.txt entries.
- You MUST run CI against each of the "major versions" (2.10, 2.11, 2.12, etc) of `ansible-base/ansible-core` that the collection supports. (Usually the `HEAD` of the stable-xxx branches.)
- All CI tests MUST run against every pull request and SHOULD pass before merge.
- At least sanity tests MUST run against a commit that releases the collection; if they do not pass, the collection will NOT be released.
  - If the collection has integration/unit tests, they SHOULD run too; if they do not pass, the errors SHOULD be analyzed to decide whether they should block the release or not.
- All CI tests MUST run regularly (nightly, or at least once per week) to ensure that repositories without regular commits are tested against the latest version of ansible-test from each ansible-base/ansible-core version tested. The results from the regular CI runs MUST be checked regularly.

All of the above can be achieved by using the GitHub Action template.

To learn how to add tests to your collection, see:

- *Adding integration tests to a collection*
- *Add unit tests to a collection*

Collections and Working Groups

The collections have:

- Working group page(s) on a corresponding wiki if needed. Makes sense if there is a group of modules for working with one common entity, for example postgresql, zabbix, grafana, and so on.
- Issue for agenda (or pinboard if there are not regular meetings) as a pinned issue in the repository.

When moving modules between collections

All related entities must be moved/copied including:

- Related plugins and module_utils files (when moving, be sure it is not used by other modules, otherwise copy).
- CI and unit tests.
- Corresponding documentation fragments from `plugins/doc_fragments`.

Also:

- Change `M()`, examples, `seealso`, `extended_documentation_fragments` to use actual FQCNs in moved content and in other collections that have references to the content.
- Move all related issues, pull requests, and wiki pages.
- Look through `docs/docsite` directory of ansible-base GitHub repository (for example, using the `grep` command-line utility) to check if there are examples using the moved modules and plugins to update their FQCNs.

See Migrating content to a different collection for complete details.

Development conventions

Besides all the requirements listed in the *Conventions, tips, and pitfalls*, be sure:

- Your modules satisfy the concept of idempotency: if a module repeatedly runs with the same set of inputs, it will not make any changes on the system.
- Your modules do not query information using special `state` option values like `get`, `list`, `query`, or `info` - create new `_info` or `_facts` modules instead (for more information, refer to the Developing modules guidelines).
- `check_mode` is supported in all `*_info` and `*_facts` modules (for more information, refer to the Development conventions).

Collection Dependencies

Notation: if foo.bar has a dependency on baz.bam, we say that baz.bam is the collection *depended on*, and foo.bar is the *dependent collection*.

- Collection dependencies must have a lower bound on the version which is at least 1.0.0.
  - This means that all collection dependencies have to specify lower bounds on the versions, and these lower bounds should be stable releases, and not versions of the form 0.x.y.
  - When creating new collections where collection dependencies are also under development, you need to watch out since Galaxy checks whether dependencies exist in the required versions:
    1. Assume that `foo.bar` depends on `foo.baz`.
    2. First release `foo.baz` as 1.0.0.
    3. Then modify `foo.bar`'s `galaxy.yml` to specify `'>=1.0.0'` for `foo.baz`.
    4. Finally release `foo.bar` as 1.0.0.
- The dependencies between collections included in Ansible must be valid. If a dependency is violated, the involved collections must be pinned so that all dependencies are valid again. This means that the version numbers from the previous release are kept or only partially incremented so that the resulting set of versions has no invalid dependencies.

- If a collection has a too strict dependency for a longer time, and forces another collection depended on to be held back, that collection will be removed from the next major Ansible release. What "longer time" means depends on when the next Ansible major release happens. If a dependent collection prevents a new major version of a collection it depends on to be included in the next major Ansible release, the dependent collection will be removed from that major release to avoid blocking the collection being depended on.
- We strongly suggest that collections also test against the `main` branches of their dependencies to ensure that incompatibilities with future releases of these are detected as early as possible and can be resolved in time to avoid such problems. Collections depending on other collections must understand that they bear the risk of being removed when they do not ensure compatibility with the latest releases of their dependencies.
- Collections included in Ansible must not depend on other collections except if they satisfy one of the following cases:
  1. They have a loose dependency on one (or more) major versions of other collections included in Ansible. For example, `ansible.netcommon: >=1.0.0`, or `ansible.netcommon: >=2.0.0, <3.0.0`. In case the collection depended on releases a new major version outside of this version range that will be included in the next major Ansible release, the dependent collection will be removed from the next major Ansible release. The cut-off date for this is feature freeze.
  2. They are explicitly being allowed to do so by the Steering Committee.

Examples

1. `community.foo 1.2.0` has a dependency on `community.bar >= 1.0.0, < 1.3.0`.
   - Now `community.bar` creates a new release `1.3.0`. When `community.foo` does not create a new release with a relaxed dependency, we have to include `community.bar 1.2.x` in the next Ansible release despite `1.3.0` being available.
   - If `community.foo` does not relax its dependency on `community.bar` for some time, `community.foo` will be removed from the next Ansible major release.
   - Unfortunately `community.bar` has to stay at `1.2.x` until either `community.foo` is removed (in the next major release), or loosens its requirements so that newer `community.bar 1.3.z` releases can be included.
2. `community.foonetwork` depends on `ansible.netcommon >= 2.0.0, <3.0.0`.
   - `ansible.netcommon 4.0.0` is released during this major Ansible release cycle.
   - `community.foonetwork` either releases a new version before feature freeze of the next major Ansible release that allows depending on all `ansible.netcommon 4.x.y` releases, or it will be removed from the next major Ansible release.

Requirements for collections to be included in the Ansible Package

To be included in the *ansible* package, collections must meet the following criteria:

- *Development conventions*.
- Collection requirements (this document).
  - The Collection Inclusion Criteria Checklist covers most of the criteria from this document.
- *Ansible documentation format* and the style guide.
- To pass the Ansible sanity tests.
- To have unit according to the corresponding sections of this document.

Other requirements

- After content is moved out of another currently included collection such as `community.general` or `community.network` OR a new collection satisfies all the requirements, add the collection to the `ansible.in` file in a corresponding directory of the ansible-build-data repository.

1.13.7 Guidelines for collection maintainers

Thank you for being a community collection maintainer. This guide offers an overview of your responsibilities as a maintainer along with resources for additional information. The Ansible community hopes that you will find that maintaining a collection is as rewarding for you as having the collection content is for the wider community.

Maintainer responsibilities

An Ansible collection maintainer is a contributor trusted by the community who makes significant and regular contributions to the project and who has shown themselves as a specialist in the related area. Collection maintainers have *extended permissions* in the collection scope.

Ansible collection maintainers provide feedback, responses, or actions on pull requests or issues to the collection(s) they maintain in a reasonably timely manner. They can also update the contributor guidelines for that collection, in collaboration with the Ansible community team and the other maintainers of that collection.

In general, collection maintainers:

- Act in accordance with the *Community Code of Conduct*.
- Subscribe to the collection repository they maintain (click *Watch* > *All activity* in GitHub).
- Keep README, development guidelines, and other general collections *Maintaining good collection documentation* relevant.
- Review and commit changes made by other contributors.
- *Backport* changes to stable branches.
- Address or assign issues to appropriate contributors.
- *Release collections*.
- Ensure that collections adhere to the *Ansible community package collections requirements*.
- Track changes announced in News for collection contributors and maintainers and update a collection in accordance with these changes.
- Subscribe and submit news to the Bullhorn newsletter.
- *Build a healthy community* to increase the number of active contributors and maintainers around collections.

- Revise these guidelines to improve the maintainer experience for yourself and others.

Multiple maintainers can divide responsibilities among each other.

How to become a maintainer

A person interested in becoming a maintainer and satisfying the *requirements* may either self-nominate or be nominated by another maintainer.

To nominate a candidate, create a GitHub issue in the relevant collection repository. If there is no response, the repository is not actively maintained, or the current maintainers do not have permissions to add the candidate, please create the issue in the ansible/community repository.

Communicating as a collection maintainer

> Maintainers MUST subscribe to the "Changes impacting collection contributors and maintainers" GitHub repo and the Bullhorn newsletter. If you have something important to announce through the newsletter (for example, recent releases), see the Bullhorn's wiki page to learn how.

Collection contributors and maintainers should also communicate through:

- *Real-time chat* appropriate to their collection, or if none exists, the general community and developer chat channels
- Mailing lists such as ansible-announce and ansible-devel
- Collection project boards, issues, and GitHub discussions in corresponding repositories
- Quarterly Contributor Summits.
- Ansiblefest and local meetups.

See *Communicating with the Ansible community* for more details on these communication channels.

Establishing working group communication

Working groups depend on efficient, real-time communication. Project maintainers can use the following techniques to establish communication for working groups:

- Find an existing *Working groups* that is similar to your project and join the conversation.
- Request a new working group for your project.
- Create a public chat for your working group or ask the community team.
- Provide working group details and links to chat rooms in the contributor section of your project `README.md`.
- Encourage contributors to join the chats and add themselves to the working group.

See the *Communication guide* to learn more about real-time chat.

Community Topics

The Community and the Steering Committee asynchronously discuss and vote on the Community Topics which impact the whole project or its parts including collections and packaging.

Share your opinion and vote on the topics to help the community make the best decisions.

Contributor Summits

The quarterly Ansible Contributor Summit is a global event that provides our contributors a great opportunity to meet each other, communicate, share ideas, and see that there are other real people behind the messages on Matrix or Libera Chat IRC, or GitHub. This gives a sense of community. Watch the Bullhorn newsletter for information when the next contributor summit, invite contributors you know, and take part in the event together.

Weekly community Matrix/IRC meetings

The Community and the Steering Committee come together at weekly meetings in the `#ansible-community` Libera.Chat IRC channel or in the bridged #community:ansible.com room on Matrix to discuss important project questions. Join us! Here is our schedule.

Expanding the collection community

Note: If you discover good ways to expand a community or make it more robust, edit this section with your ideas to share with other collection maintainers.

Here are some ways you can expand the community around your collection:

- Give *newcomers a positive first experience*.
- Invite contributors to join *real-time chats* related to your project.
- Have *good documentation* with guidelines for new contributors.
- Make people feel welcome personally and individually.
- Use labels to show easy fixes and leave non-critical easy fixes to newcomers and offer to mentor them.
- Be responsive in issues, PRs and other communication.
- Conduct PR days regularly.
- Maintain a zero-tolerance policy towards behavior violating the *Community Code of Conduct*.
- Put information about how people can register code of conduct violations in your `README` and `CONTRIBUTING` files.
- Include quick ways contributors can help and other documentation in your `README`.
- Add and keep updated the `CONTRIBUTORS` and `MAINTAINERS` files.
- Create a pinned issue to announce that the collection welcomes new maintainers and contributors.
- Look for new maintainers among active contributors.
- Announce that your collection welcomes new maintainers.
- Take part and congratulate new maintainers in Contributor Summits.

Encouraging new contributors

Easy-fix items are the best way to attract and mentor new contributors. You should triage incoming issues to mark them with labels such as `easyfix`, `waiting_on_contributor`, and `docs`. where appropriate. Do not fix these trivial non-critical bugs yourself. Instead, mentor a person who wants to contribute.

For some easy-fix issues, you could ask the issue reporter whether they want to fix the issue themselves providing the link to a quick start guide for creating PRs.

Conduct pull request days regularly. You could plan PR days, for example, on the last Friday of every month when you and other maintainers go through all open issues and pull requests focusing on old ones, asking people if you can help, and so on. If there are pull requests that look abandoned (for example, there is no response on your help offers since the previous PR day), announce that anyone else interested can complete the pull request.

Promote active contributors satisfying *requirements* to maintainers. Revise contributors' activity regularly.

If your collection found new maintainers, announce that fact in the Bullhorn newsletter and during the next Contributor Summit congratulating and thanking them for the work done. You can mention all the people promoted since the previous summit. Remember to invite the other maintainers to the Summit in advance.

Some other general guidelines to encourage contributors:

- Welcome the author and thank them for the issue or pull request.
- If there is a non-crucial easy-fix bug reported, politely ask the author to fix it themselves providing a link to *Creating your first collection pull request*.
- When suggesting changes, try to use questions, not statements.
- When suggesting mandatory changes, do it as politely as possible providing documentation references.
- If your suggestion is optional or a matter of personal preference, please say it explicitly.
- When asking for adding tests or for complex code refactoring, say that the author is welcome to ask for clarifications and help if they need it.
- If somebody suggests a good idea, mention it or put a thumbs up.
- After merging, thank the author and reviewers for their time and effort.

See the *Review checklist for collection PRs* for a list of items to check before you merge a PR.

Maintaining good collection documentation

Maintainers look after the collection documentation to ensure it matches the *Ansible documentation style guide*. This includes keeping the following documents accurate and updated regularly:

- Collection module and plugin documentation that adheres to the *Ansible documentation format*.
- Collection user guides that follow the *Collection documentation format*.
- Repository files that includes at least a `README` and `CONTRIBUTING` file.

A good `README` includes a description of the collection, a link to the *Community Code of Conduct*, and details on how to contribute or a pointer to the `CONTRIBUTING` file. If your collection is a part of Ansible (is shipped with Ansible package), highlight that fact at the top of the collection's `README`.

> The `CONTRIBUTING` file includes all the details or links to the details on how a new or continuing contributor can contribute to this collection. The `CONTRIBUTING` file should include:

- Information or links to new contributor guidelines, such as a quick start on opening PRs.
- Information or links to contributor requirements, such as unit and integration test requirements.

You can optionally include a `CONTRIBUTORS` and `MAINTAINERS` file to list the collection contributors and maintainers.

Backporting and Ansible inclusion

Each collection community can set its own rules and workflow for managing pull requests, bug reports, documentation issues, and feature requests, as well as adding and replacing maintainers. Maintainers review and merge pull requests following the:

- *Community Code of Conduct*
- *Maintainer responsibilities*
- *Committer guidelines*
- *PR review checklist*

There can be two kinds of maintainers: *Collection maintainers* and *Module maintainers*.

Collection maintainers

Collection-scope maintainers are contributors who have the `write` or higher access level in a collection. They have commit rights and can merge pull requests, among other permissions.

When a collection maintainer considers a contribution to a file significant enough (for example, fixing a complex bug, adding a feature, providing regular reviews, and so on), they can invite the author to become a module maintainer.

Module maintainers

Module-scope maintainers exist in collections that have the collection bot, for example, community.general and community.network.

Being a module maintainer is the stage prior to becoming a collection maintainer. Module maintainers are contributors who are listed in `.github/BOTMETA.yml`. The scope can be any file (for example, a module or plugin), directory, or repository. Because in most cases the scope is a module or group of modules, we call these contributors module maintainers. The collection bot notifies module maintainers when issues/pull requests related to files they maintain are created.

Module maintainers have indirect commit rights implemented through the collection bot. When two module maintainers comment with the keywords `shipit`, `LGTM`, or `+1` on a pull request which changes a module they maintain, the collection bot merges the pull request automatically.

For more information about the collection bot and its interface, see to the Collection bot overview.

Releasing a collection

Collection maintainers are responsible for releasing new versions of a collection. Generally, releasing a collection consists of:

1. Planning and announcement.
2. Generating a changelog.
3. Creating a release git tag and pushing it.
4. Automatically publishing the release tarball on Ansible Galaxy through the Zuul dashboard.
5. Final announcement.

6. Optionally, file a request to include a new collection into the Ansible package.

See *Releasing collections* for details.

Backporting

Collection maintainers backport merged pull requests to stable branches following the semantic versioning and release policies of the collections.

The manual backport process is similar to the *ansible-core backporting guidelines*.

For convenience, backporting can be implemented automatically using GitHub bots (for example, with the Patchback app) and labeling as it is done in community.general and community.network.

Including a collection in Ansible

If a collection is not included in Ansible (not shipped with Ansible package), maintainers can submit the collection for inclusion by creating a discussion under the ansible-collections/ansible-inclusion repository. For more information, see the repository's README, and the *Ansible community package collections requirements*.

Stepping down as a collection maintainer

Times change, and so may your ability to continue as a collection maintainer. We ask that you do not step down silently.

If you feel you don't have time to maintain your collection anymore, you should:

- Inform other maintainers about it.
- If the collection is under the `ansible-collections` organization, also inform relevant *Real-time chat*, the `community` chat channels on IRC or matrix, or by email `ansible-community@redhat.com`.
- Look at active contributors in the collection to find new maintainers among them. Discuss the potential candidates with other maintainers or with the community team.
- If you failed to find a replacement, create a pinned issue in the collection, announcing that the collection needs new maintainers.
- Make the same announcement through the Bullhorn newsletter.
- Please be around to discuss potential candidates found by other maintainers or by the community team.

Remember, this is a community, so you can come back at any time in the future.

Releasing collections

Collection maintainers release all supported stable versions of the collections regularly, provided that there have been enough changes merged to release.

Preparing to release a collection

The collections under the ansible-collections organization follow semantic versioning when releasing. See *Collection versioning and deprecation* for details.

To prepare for a release, a collection must have:

- A publicly available policy of releasing, versioning, and deprecation. This can be, for example, written in its README or in a dedicated pinned issue.
- A pinned issue when its release managers inform the community about planned or completed releases. This can be combined with the release policy issue mentioned above.
- A *changelog*.
- Releases of the collection tagged in the collection's repository.
- CI pipelines up and running. This can be implemented by using GitHub Actions, Azure Pipelines, Zuul.
- All CI tests running against a commit that releases the collection. If they do not pass, the collection MUST NOT be released.

See *Including a collection in Ansible* if you plan on adding a new collection to the Ansible package.

Note: Your collection must pass `ansible-test sanity` tests. See *Testing collections* for details.

Collection versioning and deprecation

Note: Collections MUST adhere to semantic versioning.

To preserve backward compatibility for users, every Ansible minor version series (5.1.x, 5.2.x, and so on) will keep the major version of a collection constant. For example, if Ansible 5.0.0 includes `community.general` 4.0.2, then each Ansible 5.X.x release will include the latest `community.general` 4.y.z release available at build time. Ansible 5.x.x will **never** include a `community.general` 5.y.x release, even if it is available. Major collection version changes will be included in the next Ansible major release (6.0.0 in this case). Ensure that the current major release of your collection included in 6.0.0 receives at least bugfixes as long as new Ansible 6.X.X releases are produced.

Since new minor releases are included, you can include new features, modules and plugins. You must make sure that you do not break backwards compatibility. See semantic versioning. for more details. This means in particular:

- You can fix bugs in **patch** releases but not add new features or deprecate things.
- You can add new features and deprecate things in **minor** releases, but not remove things or change behavior of existing features.
- You can only remove things or make breaking changes in **major** releases.

Ensure that if a deprecation is added in a collection version that is included in 5.x.y, the removal itself will only happen in a collection version included in 7.0.0 or later. Ensure that the policy of releasing, versioning, and deprecation is announced to contributors and users in some way. For an example of how to do this, see the announcement in community.general. You could also do this in the collection README file.

Collection changelogs

Collections MUST include a changelog. To give a consistent feel for changelogs across collections and ensure changelogs exist for collections included in the `ansible` package, we suggest you use antsibull-changelog to maintain and generate this.

Before releasing, verify the following for your changelogs:

- All merged pull requests since the last release, except ones related to documentation and new modules/plugins, have *changelog fragments*.
- New module and plugin pull requests, except jinja2 test and filter plugins, do **not** need a changelog fragment, they are auto-detected by the changelog generator by their `version_added` value.
- All the fragments follow the *changelog entry format*.

Options for releasing a collection

There are several approaches on how to release a collection. If you are not aware of which approach to use, ask in the `#ansible-community` IRC channel or the `community` Matrix channel.

This section assumes that publishing the collection is done with Zuul and that antsibull-changelog is used for the changelog.

Releasing without release branches

Use releasing without release branches when:

- There are no prior major releases of the collection.
- There are no breaking changes introduced since the `1.0.0` release of the collection.

See *Releasing collections without release branches* for details.

When there is a need to introduce breaking changes, you can switch to the next approach.

Hybrid approach

In this approach, releases for the current major version are made from the `main` branch, while new releases for older major versions are made from release branches for these versions.

Releasing with release branches

Use releasing with release branches when breaking changes have been introduced. This approach is usually only used by the large community collections, `community.general` and `community.network`.

See *Releasing collections with release branches* for details.

Releasing collections without release branches

Since no release branches are used, this section does not distinguish between releasing a major, minor, or patch version.

Release planning and announcement

1. Examine the collection to determine if there are merged changes to release.
2. According to the changes made, choose an appropriate release version number. Keep in mind that the collections must follow the semantic versioning rules. See *Collection versioning and deprecation* for details.
3. Announce your intention to release the collection in a corresponding pinned release issue or community pinboard of the collection and in the `community` *Matrix/IRC channel*.

Creating the release branch

1. Ensure you are in a default branch in your local fork. We use `main` in the following examples.

   ```
   git status
   git checkout main     # if needed
   ```

2. Update your local fork:

   ```
   git pull --rebase upstream main
   ```

3. Checkout a new release branch from the default branch:

   ```
   git checkout -b release_branch
   ```

4. Ensure the `galaxy.yml` contains the correct release version number.

Generating the changelog

1. Add a changelog fragment `changelogs/fragments/X.Y.Z.yml` with content:

   ```
   release_summary: |-
     Write some text here that should appear as the release summary for this␣
   ↪version.
     The format is reStructuredText, but not a list as for regular changelog␣
   ↪fragments.
     This text will be inserted into the changelog.
   ```

   For example:

```
release_summary: |-
  This is the minor release of the ``community.mysql`` collection.
  This changelog contains all changes to the modules and plugins in this␣
↪collection
  that have been made after the previous release.
```

2. If the content was recently moved from another collection (for example, migrating a module from one collection to another), ensure you have all related changelog fragments in the `changelogs/fragments` directory. If not, copy them previously.

3. Run `antsibull-changelog release --reload-plugins` . This package should previously be installed with `pip install antsibull-changelog`.

4. Verify that the `CHANGELOG.rst` looks as expected.

5. Commit and push changes to the `CHANGELOG.rst` and `changelogs/changelog.yaml`, and potentially deleted/archived fragments to the `origin` repository's `release_branch`.

```
git commit -a -m "Release VERSION commit"
git push origin release_branch
```

6. Create a pull request in the collection repository. If CI tests pass, merge it.

7. Checkout the default branch and pull the changes:

```
git checkout main
git pull --rebase upstream main
```

Publish the collection

1. Add an annotated tag to the release commit with the collection version. Pushing this tag to the `upstream` repository will make Zuul publish the collection on Ansible Galaxy.

```
git tag -n    # see current tags and their comments
git tag -a NEW_VERSION -m "comment here"    # the comment can be, for example,␣
↪ "community.postgresql: 1.2.0"
git push upstream NEW_VERSION
```

2. Wait until the new version is published on the collection's Ansible Galaxy page. It will appear in a list of tarballs available to download.

3. Update the version in the `galaxy.yml` file to the next **expected** version. Add, commit, and push to the `upstream`'s default branch.

4. Add a GitHub release for the new tag. Title should be the version and content `See https://github.com/ansible-collections/community.xxx/blob/main/CHANGELOG.rst for all changes`.

5. Announce the release through the Bullhorn Newsletter issue.

6. Announce the release in the pinned release issue/community pinboard of the collection mentioned in step 3 and in the `community` *Matrix/IRC channel*.

Releasing collections with release branches

Collections MUST follow the semantic versioning rules. See *Releasing collections* for high-level details.

Release planning and announcement

1. Announce your intention to release the collection in a corresponding pinned release issue/community pinboard of the collection and in the `#ansible-community` Matrix/IRC channel. Repeat the announcement in any other dedicated channels if they exist.
2. Ensure all the other repository maintainers are informed about the time of the following release.

Releasing major collection versions

The new version is assumed to be `X.0.0`.

1. Make sure that `galaxy.yml` contains the correct version number `X.0.0`. If that is not the case, create a PR to update it. This will make sanity tests fail for all deprecations that have to be removed in `X.0.0`, so this is potentially a lot of work and should have been done weeks before the major release.
2. Check the collection for deprecations that are planned for removal in the major release that were not reported by the sanity tests. Use past changelogs or run ``grep -r `X.0.0` plugins/`` in the repository.
3. If you are going to release the `community.general` and `community.network` collections, create a new `backport-X` label in the corresponding repositories. Copy the styles and descriptions from the corresponding existing labels.
4. Ensure you are in a default branch in your local fork. These examples use `main`.

   ```
   git status
   git checkout main     # if needed
   ```

5. Update your local fork:

   ```
   git pull --rebase upstream main
   ```

Creating the release branch

1. Create a branch `stable-X`. Replace `X` with a correct number and push it to the **upstream** repository, NOT to the `origin`.:

   ```
   git branch stable-X main
   git push upstream stable-X
   ```

2. Create and checkout to another branch from the `main` branch:

   ```
   git checkout -b update_repo
   ```

3. Update the version in `galaxy.yml` in the branch to the next **expected** version, for example, `X.1.0`.

Creating the changelogs

1. Replace `changelogs/changelog.yml` with:

   ```
   ancestor: X.0.0
   releases: {}
   ```

2. Remove all changelog fragments from `changelogs/fragments/`. Removing the changelog fragments ensures that every major release has a changelog describing changes since the last major release.
3. Add and commit all the changes made. Push the branch to the `origin` repository.
4. Create a pull request in the collection repository. If CI tests pass, merge the pull request since the `main` branch is expecting changes for the next minor/major versions
5. Switch to the `stable-X` branch.
6. In the `stable-X` branch, verify that `galaxy.yml` contains the correct version number `X.0.0`.
7. In the `stable-X` branch, ensure that `changelogs/changelog.yml` contains a correct ancestor's version:

   ```
   ancestor: X-1.0.0
   releases: {}
   ```

8. In the `stable-X` branch, add a changelog fragment `changelogs/fragments/X.0.0.yml` with the content:

   ```
   release_summary: |-
     Write some text here that should appear as the release summary for this␣
   ↪version.
     The format is reStructuredText, but not a list as for regular changelog␣
   ↪fragments.
     This text will be inserted into the changelog.
   ```

   For example:

   ```
   release_summary: This is release 2.0.0 of ``community.foo``, released on YYYY-
   ↪MM-DD.
   ```

9. In the `stable-X` branch, generate the changelogs:

   ```
   antsibull-changelog release --cummulative-release
   ```

10. In the `stable-X` branch, verify that the `CHANGELOG.rst` looks as expected.

11. In the `stable-X` branch, update `README.md` so that the changelog link points to `/tree/stable-X/` and no longer to `/tree/main/`, and change badges respectively, for example, in case of AZP, add `?branchName=stable-X` to the AZP CI badge (https://dev.azure.com/ansible/community.xxx/_apis/build/status/CI?branchName=stable-X).
12. In the `stable-X` branch, add, commit, and push changes to `README.md`, `CHANGELOG.rst` and `changelogs/changelog.yaml`, and potentially deleted/archived fragments to the **upstream** repository, NOT to the `origin`.

Publishing the collection

1. In the `stable-X` branch, add an annotated tag to the last commit with the collection version `X.0.0`. Pushing this tag to the `upstream` repository will make Zuul publish the collection on Ansible Galaxy.

```
git tag -n    # see current tags and their comments
git tag -a NEW_VERSION -m "comment here"    # the comment can be, for example,
↪"community.foo: 2.0.0"
git push upstream NEW_VERSION
```

2. If the collection uses Zuul for publishing its releases, wait until the new version is published on the collection's Ansible Galaxy page. It will appear in a list of tarballs available to download.
3. If the release tarball did not appear within several hours after pushing the tag, try to re-tag the release commit and push the tag again. In the `stable-X` branch being at the release commit:

```
git tag --delete NEW_VERSION
git push upstream :NEW_VERSION
git tag -a NEW_VERSION -m "comment here"    # the comment can be, for example,
↪"community.foo: 2.0.0"
git push upstream NEW_VERSION
```

4. Add a GitHub release for the new tag. The title should be the version and content, such as `- See https://github.com/ansible-collections/community.xxx/blob/stable-X/CHANGELOG.rst for all changes`.
5. Announce the release through the Bullhorn Newsletter.
6. Announce the release in the pinned release issue/community pinboard of the collection and in the `#ansible-community` Matrix/Libera.Chat IRC channel.
7. In the `stable-X` branch, update the version in `galaxy.yml` to the next **expected** version, for example, `X.1.0`. Add, commit and push to the **upstream** repository.

Releasing minor collection versions

The new version is assumed to be `X.Y.0`. All changes that should go into it are expected to be previously backported from the default branch to the `stable-X` branch.

Creating the changelogs

1. In the `stable-X` branch, make sure that `galaxy.yml` contains the correct version number `X.Y.0`. If not, update it.
2. In the `stable-X` branch, add a changelog fragment `changelogs/fragments/X.Y.0.yml` with content:

```
release_summary: |-
  Write some text here that should appear as the release summary for this␣
↪version.
  The format is reStructuredText, but not a list as for regular changelog␣
↪fragments.
  This text will be inserted into the changelog.
```

3. In the `stable-X` branch, run:

```
antsibull-changelog release
```

4. In the `stable-X` branch, verify that `CHANGELOG.rst` looks as expected.
5. In the `stable-X` branch, add, commit, and push changes to `CHANGELOG.rst` and `changelogs/changelog.yaml`, and potentially deleted/archived fragments to the **upstream** repository, NOT to the origin.

Publishing the collection

1. In the `stable-X` branch, add an annotated tag to the last commit with the collection version `X.Y.0`. Pushing this tag to the `upstream` repository will make Zuul publish the collection on Ansible Galaxy.

```
git tag -n    # see current tags and their comments
git tag -a NEW_VERSION -m "comment here"    # the comment can be, for example,
↪"community.foo: 2.1.0"
git push upstream NEW_VERSION
```

2. Wait until the new version is published on the collection's Ansible Galaxy page. The published version will appear in a list of tarballs available to download.
3. Add a GitHub release for the new tag. The title should be the version and content, such as `- See https://github.com/ansible-collections/community.xxx/blob/stable-X/CHANGELOG.rst for all changes`.
4. Announce the release through the Bullhorn Newsletter.
5. Announce the release in the pinned release issue/community pinboard of the collection and in the `#ansible-community` Matrix/IRC channel. Additionally, you can announce it using GitHub's Releases system.
6. In the `stable-X` branch, update the version in `galaxy.yml` to the next **expected** version, for example, if you have released `X.1.0`, the next expected version could be `X.2.0`. Add, commit and push to the **upstream** repository.
7. Checkout to the `main` branch.
8. In the `main` branch:
   1. If more minor versions are released before the next major version, update the version in `galaxy.yml` to `X.(Y+1).0` as well. Create a dedicated pull request and merge.

2. If the next version will be a new major version, create a pull request where you update the version in `galaxy.yml` to `(X+1).0.0`. Note that the sanity tests will most likely fail since there will be deprecations with removal scheduled for `(X+1).0.0`, which are flagged by the tests.

   For every such deprecation, decide:

   - Whether to remove them now. For example you remove the complete `modules/plugins` or you remove redirects.
   - Whether to add ignore entries to the corresponding `tests/sanity/ignore-*.txt` file and create issues, for example for removed features in `modules/plugins`.

   Once the CI tests pass, merge the pull request. Make sure that this pull request is merged not too much later after the release for `version_added` sanity tests not to expect the wrong version for the new feature pull request.

Note: It makes sense to already do some removals in the days before the release. These removals must happen in the main branch and must not be backported.

Releasing patch versions

The new version is assumed to be `X.Y.Z`, and the previous patch version is assumed to be `X.Y.z` with `z < Z`. `z` is frequently``0`` since patch releases are uncommon.

Releasing when more minor versions are expected

1. Checkout the `X.Y.z` tag.
2. Update `galaxy.yml` so that the version is `X.Y.Z`. Add and commit.
3. Cherry-pick all changes from `stable-X` that were added after `X.Y.z` and should go into `X.Y.Z`.
4. Add a changelog fragment `changelogs/fragments/X.Y.Z.yml` with content:

```
release_summary: |-
  Write some text here that should appear as the release summary for this␣
↪version.
  The format is reStructuredText but not a list as for regular changelog␣
↪fragments.
  This text will be inserted into the changelog.
```

   Add to git and commit.

5. Generate the changelogs.

```
antsibull-changelog release
```

6. Verify that `CHANGELOG.rst` looks as expected.
7. Add and commit changes to `CHANGELOG.rst` and `changelogs/changelog.yaml`, and potentially deleted/archived fragments.

Publishing the collection

1. Add an annotated tag to the last commit with the collection version `X.Y.Z`. Pushing this tag to the `upstream` repository will make Zuul publish the collection on Ansible Galaxy.

```
git tag -n    # see current tags and their comments
git tag -a NEW_VERSION -m "comment here"    # the comment can be, for example,
↪"community.foo: 2.1.1"
git push upstream NEW_VERSION
```

2. Wait until the new version is published on the collection's Ansible Galaxy page. It will appear in a list of tarballs available to download.

3. Add a GitHub release for the new tag. The title should be the version and content, such as `- See https://github.com/ansible-collections/community.xxx/blob/stable-X/CHANGELOG.rst for all changes`.

Note: The data for this release is only contained in a tag, and not in a branch, in particular not in `stable-X`. This is deliberate, since the next minor release `X.(Y+1).0` already contains the changes for `X.Y.Z` as well, since these were cherry-picked from `stable-X`.

4. Announce the release through the Bullhorn Newsletter.

5. Announce the release in the pinned release issue/community pinboard of the collection and in the `#ansible-community` *Matrix/IRC channel* <*https://docs.ansible.com/ansible/devel/community/communication.html#real-time-chat*>.

Releasing when no more minor versions are expected

1. In the `stable-X` branch, make sure that `galaxy.yml` contains the correct version number `X.Y.Z`. If not, update it!

2. In the `stable-X` branch, add a changelog fragment `changelogs/fragments/X.Y.Z.yml` with content:

```
release_summary: |-
  Write some text here that should appear as the release summary for this␣
↪version.
  The format is reStructuredText, but not a list as for regular changelog␣
↪fragments.
  This text will be inserted into the changelog.
```

3. Generate the changelogs in the `stable-X` branch.

```
antsibull-changelog release
```

4. In the `stable-X` branch, verify that `CHANGELOG.rst` looks as expected.

5. In the `stable-X` branch, add, commit, and push changes to `CHANGELOG.rst` and `changelogs/changelog.yaml`, and potentially deleted/archived fragments to the **upstream** repository, NOT to the origin.

Publishing the collection

1. In the `stable-X` branch, add an annotated tag to the last commit with the collection version `X.Y.Z`. Pushing this tag to the `upstream` repository will make Zuul publish the collection on Ansible Galaxy.

```
git tag -n    # see current tags and their comments
git tag -a NEW_VERSION -m "comment here"    # the comment can be, for example,
↪"community.foo: 2.1.1"
git push upstream NEW_VERSION
```

2. Wait until the new version is published on the collection's Ansible Galaxy page. It will appear in a list of tarballs available to download.
3. Add a GitHub release for the new tag. Title should be the version and content, such as: `See https://github.com/ansible-collections/community.xxx/blob/stable-X/CHANGELOG.rst for all changes`.
4. Announce the release through the Bullhorn Newsletter.
5. Announce the release in the pinned issue/community pinboard of the collection and in the `#ansible-community` Matrix/IRC channel.

In addition to the information here, module maintainers should be familiar with:

- *General Ansible community development practices*
- Documentation on *module development*

1.13.8 Contributing to Ansible-maintained Collections

The Ansible team welcomes community contributions to the collections maintained by Red Hat Ansible Engineering. This section describes how you can open issues and create PRs with the required testing before your PR can be merged.

Ansible-maintained collections

The following table shows:

- **Ansible-maintained collection** - Click the link to the collection on Galaxy, then click the `repo` button in Galaxy to find the GitHub repository for this collection.
- **Related community collection** - Collection that holds community-created content (modules, roles, and so on) that may also be of interest to a user of the Ansible-maintained collection. You can, for example, add new modules to the community collection as a technical preview before the content is moved to the Ansible-maintained collection.
- **Sponsor** - Working group that manages the collections. You can join the meetings to discuss important proposed changes and enhancements to the collections.
- **Test requirements** - Testing required for any new or changed content for the Ansible-maintained collection.
- **Developer details** - Describes whether the Ansible-maintained collection accepts direct community issues and PRs for existing collection content, as well as more specific developer guidelines based on the collection type.

Note: * A ✓ under **Open to PRs** means the collection welcomes GitHub issues and PRs for any changes to existing collection content (plugins, roles, and so on).

** Integration tests are required and unit tests are welcomed but not required for the AWS collections. An exception to this is made in cases where integration tests are logistically not feasible due to external requirements. An example of this is AWS Direct Connect, as this service can not be functionally tested without the establishment of network peering connections. Unit tests are therefore required for modules that interact with AWS Direct Connect. Exceptions to `amazon.aws` must be approved by Red Hat, and exceptions to `community.aws` must be approved by the AWS community.

*** `ansible.netcommon` contains all foundational components for enabling many network and security *platform* collections. It contains all connection and filter plugins required, and installs as a dependency when you install the platform collection.

**** Unit tests for Windows PowerShell modules are an exception to testing, but unit tests are valid and required for the remainder of the collection, including Ansible-side plugins.

Deciding where your contribution belongs

We welcome contributions to Ansible-maintained collections. Because these collections are part of a downstream supported Red Hat product, the criteria for contribution, testing, and release may be higher than other community collections. The related community collections (such as `community.general` and `community.network`) have less-stringent requirements and are a great place for new functionality that may become part of the Ansible-maintained collection in a future release.

The following scenarios use the `arista.eos` to help explain when to contribute to the Ansible-maintained collection, and when to propose your change or idea to the related community collection:

1. You want to fix a problem in the `arista.eos` Ansible-maintained collection. Create the PR directly in the arista.eos collection GitHub repository. Apply all the *merge requirements*.
2. You want to add a new Ansible module for Arista. Your options are one of the following:
   - Propose a new module in the `arista.eos` collection (requires approval from Arista and Red Hat).
   - Propose a new collection in the `arista` namespace (requires approval from Arista and Red Hat).
   - Propose a new module in the `community.network` collection (requires network community approval).
   - Place your new module in a collection in your own namespace (no approvals required).

Most new content should go into either a related community collection or your own collection first so that is well established in the community before you can propose adding it to the `arista` namespace, where inclusion and maintenance criteria are much higher.

Requirements to merge your PR

Your PR must meet the following requirements before it can merge into an Ansible-maintained collection:

1. The PR is in the intended scope of the collection. Communicate with the appropriate Ansible sponsor listed in the *Ansible-maintained collection table* for help.
2. For network and security domains, the PR follows the *resource module development principles*.
3. Passes *sanity tests and tox*.
4. Passes unit, and integration tests, as listed in the *Ansible-maintained collection table* and described in *Resource module integration tests*.
5. Follows Ansible guidelines. See *Should you develop a module?* and *Developing collections*.
6. Addresses all review comments.
7. Includes an appropriate *changelog*.

1.13.9 Ansible Community Steering Committee

This section focuses on the guidelines and membership of the Ansible Community Steering Committee.

Steering Committee mission and responsibilities

The Steering Committee mission is to provide continuity, guidance, and suggestions to the Ansible community to ensure the delivery and high quality of the Ansible package. In addition, the committee helps decide the technical direction of the Ansible project. It is responsible for approving new proposals and policies in the community, package, and community collections world, new community collection-inclusion requests, and other technical aspects regarding inclusion and packaging. The Committee should reflect the scope and breadth of the Ansible community.

Steering Committee responsibilities

The Committee:

- Designs policies and procedures for the community collections world.
- Votes on approval changes to established policies and procedures.
- Reviews community collections for compliance with the policies.
- Helps create and define roadmaps for our deliverables such as the `ansible` package, major community collections, and documentation.
- Reviews community collections submitted for inclusion in the Ansible package and decides whether to include them or not.
- Review other proposals of importance that need the Committee's attention and provide feedback.

Current Steering Committee members

The following table lists the current Steering Committee members. See *Steering Committee past members* for a list of past members.

Table 1: Current Steering committee members

Name	GitHub	Start year
Alexei Znamensky	russoz	2022
Alicia Cozine	acozine	2021
Andrew Klychkov	Andersson007	2021
Brad Thornton	cidrblock	2021
Brian Scholer	briantist	2022
Dylan Silva	thaumos	2021
Felix Fontein	felixfontein	2021
James Cassell	jamescassell	2021
John Barker	gundalow	2021
Mario Lenz	mariolenz	2022
Markus Bergholz	markuman	2022
Maxwell G	gotmax23	2022
Sorin Sbarnea	ssbarnea	2021

John Barker (gundalow) has been elected by the Committee as its *Chairperson*.

Committee members are selected based on their active contribution to the Ansible Project and its community. See *Steering Committee membership guidelines* to learn details.

Creating new policy proposals & inclusion requests

The Committee uses the community-topics repository to asynchronously discuss with the Community and vote on Community topics in corresponding issues.

You can create a new issue in the community-topics repository as a discussion topic if you want to discuss an idea that impacts any of the following:

- Ansible Community
- Community collection best practices and requirements
- Community collection inclusion policy
- The Community governance
- Other proposals of importance that need the Committee's or overall Ansible community attention

To request changes to the inclusion policy and collection requirements:

1. Submit a new pull request to the ansible-collections/overview repository.
2. Create a corresponding issue containing the rationale behind these changes in the community-topics repository repository.

To submit new collections for inclusion into the Ansible package:

- Submit the new collection inclusion requests through a new discussion in the ansible-inclusion repository.

Depending on a topic you want to discuss with the Community and the Committee, as you prepare your proposal, please consider the requirements established by:

- *Community Code of Conduct*.
- *Ansible community package collections requirements*.
- Ansible Collection Inclusion Checklist.

Community topics workflow

The Committee uses the Community-topics workflow to asynchronously discuss and vote on the community-topics.

The quorum, the minimum number of Committee members who must vote on a topic in order for a decision to be officially made, is half of the whole number of the Committee members. If the quorum number contains a fractional part, it is rounded up to the next whole number. For example, if there are thirteen members currently in the committee, the quorum will be seven.

Votes must always have "no change" as an option.

In case of equal numbers of votes for and against a topic, the chairperson's vote will break the tie. For example, if there are six votes for and six votes against a topic, and the chairperson's vote is among those six which are for the topic, the final decision will be positive. If the chairperson has not voted yet, other members ask them to vote.

For votes with more than two options, one choice must have at least half of the votes. If two choices happen to both have half of the votes, the chairperson's vote will break the tie. If no choice has at least half of the votes, the vote choices have to be adjusted so that a majority can be found for a choice in a new vote.

Community topics triage

The Committee conducts a triage of community topics periodically (every three to six months).

The triage goals are:

- Sparking interest for forgotten topics.
- Identifying and closing irrelevant topics, for example, when the reason of the topic does not exist anymore or the topic is out of the Committee responsibilities scope.
- Identifying and closing topics that the Community are not interested in discussing. As indicators, it can be absence of comments or no activity in comments, at least, for the last six months.
- Identifying and closing topics that were solved and implemented but not closed (in this case, such a topic can be closed on the spot with a comment that it has been implemented).
- Identifying topics that have been in pending state for a long time, for example, when it is waiting for actions from someone for several months or when the topics were solved but not implemented.

A person starting the triage:

1. Identifies the topics mentioned above.
2. Creates a special triage topic containing an enumerated list of the topics-candidates for closing.
3. Establishes a vote date considering a number of topics, their complexity and comment-history size giving the Community sufficient time to go through and discuss them.
4. The Community and the Committee vote on each topic-candidate listed in the triage topic whether to close it or keep it open.

Collection inclusion requests workflow

When reviewing community collection inclusion requests, the Committee members check if a collection adheres to the *Ansible community package collections requirements*.

1. A Committee member who conducts the inclusion review copies the Ansible community collection checklist into a corresponding discussion.
2. In the course of the review, the Committee member marks items as completed or leaves a comment saying whether the reviewer expects an issue to be addressed or whether it is optional (for example, it could be **MUST FIX:** <what> or **SHOULD FIX:** <what> under an item).
3. For a collection to be included in the Ansible community package, the collection:

- MUST be reviewed and approved by at least two persons, where at least one person is a Steering Committee member.
- For a Non-Steering Committee review to be counted for inclusion, it MUST be checked and approved by *another* Steering Committee member.
- Reviewers must not be involved significantly in development of the collection. They must declare any potential conflict of interest (for example, being friends/relatives/coworkers of the maintainers/authors, being users of the collection, or having contributed to that collection recently or in the past).

1. After the collection gets two or more Committee member approvals, a Committee member creates a community topic linked to the corresponding inclusion request. The issue's description says that the collection has been approved by two or more Committee members and establishes a date (a week by default) when the inclusion decision will be considered made. This time period can be used to raise concerns.

2. If no objections are raised up to the established date, the inclusion request is considered successfully resolved. In this case, a Committee member:

1. Declares the decision in the topic and in the inclusion request.
2. Moves the request to the `Resolved reviews` category.
3. Adds the collection to the `ansible.in` file in a corresponding directory of the ansible-build-data repository.
4. Announces the inclusion through the Bullhorn newsletter.
5. Closes the topic.

Community Working Group meetings

See the Community Working Group meeting schedule. Meeting summaries are posted in the Community Working Group Meeting Agenda issue.

Note: Participation in the Community Working Group meetings is optional for Committee members. Decisions on community topics are made asynchronously in the community-topics repository.

The meeting minutes can be found at the fedora meetbot site and the same is posted to Ansible Devel Mailing List after every meeting.

Steering Committee membership guidelines

This document describes the expectations and policies related to membership in the *Ansible Community Steering Committee* (hereinafter the Committee).

Topics:

Expectations of a Steering Committee member

As a Committee member, you agree to:

1. Abide by the *Community Code of Conduct* in all your interactions with the Community.
2. Be a Community ambassador by representing its needs within the Committee and throughout the decision making process.
3. Asynchronously participate in discussions and voting on the Community Topics.
4. Review other proposals of importance that need the Committee's attention and provide feedback.
5. Act for the sake of the Community by not promoting corporate or individual agenda during the decision making process.
6. Engage with the Community in a professional and positive manner, encourage community members to express their opinion.

Joining the Steering Committee

Eligibility

A person is eligible to become a Committee member if they have:

1. A wide knowledge of Ansible and/or its related projects.
2. Active contributions to Ansible and/or related projects in any form described in the *Ansible Collections Contributor Guide*.
3. A consent to follow the *Expectations of a Steering Committee member*.

Process

The process to join the Steering Committee consists of the following steps:

1. Any community member may nominate someone or themselves for Committee membership by contacting one of the *current Committee members*) or by sending an email to `ansible-community@redhat.com`.
2. A Committee member who receives the nomination must inform the Committee about it by forwarding the full message.
3. The vote is conducted by email. Nominees must receive a majority of votes from the present Committee members to be added to the Committee.
4. Provided that the vote result is positive, it is announced via the Bullhorn newsletter and the new member is added to the *Committee member list*.

Leaving the Steering Committee

Steering Committee members can resign voluntarily or be removed by the rest of the Steering Committee under certain circumstances. See the details below.

Voluntarily leaving the Steering Committee

A Committee member can voluntarily leave the Committee. In this case, they notify the other members, create an issue in the Community Topics repository announcing the resignation, and after that they are no longer considered Committee members.

Committee members who resign and later change their mind can rejoin the Committee by following the *Process for joining the Steering Committee*.

Involuntarily leaving the Steering Committee

A Committee member will be removed from the Committee if they:

1. Do not participate in asynchronous discussions and voting on the Community Topics for more than 3 months in a row.
2. Participate unreasonably irregularly (for example, once a month for several months). Unreasonably is defined by other Committee members considering circumstances in each particular case.
3. Violate the *Community Code of Conduct*.

Absence or irregular participation in discussing topics and votes

In case of absence or irregular participation, the involuntarily removal process consists of the following steps:

1. Another Committee member (hereinafter the initiator) contacts the person by email asking if they are still interested in fulfilling their Committee member's duties.
2. If they respond that they are not interested, the initiator asks the person to step down on their own following the *Voluntarily leaving process*.
3. If there has been no response or stepping down issue created by the person within a reasonable time, the initiator notifies other Committee members about the situation.
4. In case of agreement among the Committee about the need for removal, the initiator provides a draft of a corresponding topic's description to the Committee via email for discussion and approval.

- The topic's title is `Steering Committee member audit.`. It must not contain the person's name or other identifying information.
- The description must not contain or imply any forms of condemnation.
- It must mention that the person has been inactive for an unknown reason for the last N months and that, in accordance with the Steering Committee policies, their place should be freed for another person who can continue their great job.
- The description must mention person's achievements and thanks for their time and effort they spent serving for the Community, Committee, and the Project, and a hope that one day they will come back.

1. The initiator creates the topic in the Community Topics repository containing the description and the title from the draft.
2. The Committee members vote on the topic.

Ansible Community Code of Conduct violations

In case of the Ansible Community Code of Conduct violations, the process is the same as above except steps 1-2. Instead:

1. The initiator reports the case to the Committee via email.
2. The Committee discusses the case internally, evaluates its severity, and possible solutions.
3. If the Committee concludes that the violation is not severe, it develops a proposal to the person on how the situation can be corrected and further interactions with the Community improved.
4. A Committee representative reaches out to the person with the proposal.
5. The removal process starts if:

- The Committee decided that the severity of the violation excludes a possibility of further membership.
- The person does not respond to the proposal.
- The person explicitly rejects the proposal.

In case of starting the removal process, the topic's description in the reason's part changes correspondingly.

Chairperson

The chairperson election will happen once a year around the time of Ansible Fest. If the current chairperson has to step down early, the election happens immediately.

The process of the election consist of the following steps:

1. Members interested in being the chairperson will inform a person responsible for arranging the election about that.
2. Conduct anonymous voting somewhere.
3. Internally and publicly announce the elected candidate.

The chairperson has the following powers unlike regular members:

- The chairperson's vote breaks ties to resolve deadlocks when equal numbers of steering committee members vote for and against a community topic.

Steering Committee past members

The Ansible Community is very grateful to these amazing **nonreplaceable** people for their great service to the Community and the project!

Table 2: Steering Committee past members

Name	GitHub	Years of service
Jill Rouleau	jillr	2021-2022
Tadej Borovšak	tadeboro	2021-2022
Toshio Kuratomi	abadger	2021

We'd also like to thank our past chairpersons for their contributions to Ansible.

Table 3: Steering Committee past chairpersons

Name	GitHub	Years of service
Tadej Borovšak	tadeboro	2021-2022

1.13.10 Contributing to the Ansible Documentation

Ansible has a lot of documentation and a small team of writers. Community support helps us keep up with new features, fixes, and changes.

Improving the documentation is an easy way to make your first contribution to the Ansible project. You do not have to be a programmer, since most of our documentation is written in YAML (module documentation) or reStructuredText (rST). Some collection-level documentation is written in a subset of Markdown. If you are using Ansible, you already use YAML in your playbooks. rST and Markdown are mostly just text. You do not even need git experience, if you use the `Edit on GitHub` option.

If you find a typo, a broken example, a missing topic, or any other error or omission on this documentation website, let us know. Here are some ways to support Ansible documentation:

Editing docs directly on GitHub

For typos and other quick fixes, you can edit most of the documentation right from the site. Look at the top right corner of this page. That `Edit on GitHub` link is available on all the guide pages in the documentation. If you have a GitHub account, you can submit a quick and easy pull request this way.

Note: The source files for individual collection plugins exist in their respective repositories. Follow the link to the collection on Galaxy to find where the repository is located and any guidelines on how to contribute to that collection.

To submit a documentation PR from docs.ansible.com with `Edit on GitHub`:

1. Click on `Edit on GitHub`.
2. If you don't already have a fork of the ansible repo on your GitHub account, you'll be prompted to create one.
3. Fix the typo, update the example, or make whatever other change you have in mind.
4. Enter a commit message in the first rectangle under the heading `Propose file change` at the bottom of the GitHub page. The more specific, the better. For example, "fixes typo in my_module description". You can put more detail in the second rectangle if you like. Leave the `+label: docsite_pr` there.
5. Submit the suggested change by clicking on the green "Propose file change" button. GitHub will handle branching and committing for you, and open a page with the heading "Comparing Changes".
6. Click on `Create pull request` to open the PR template.
7. Fill out the PR template, including as much detail as appropriate for your change. You can change the title of your PR if you like (by default it's the same as your commit message). In the `Issue Type` section, delete all lines except the `Docs Pull Request` line.
8. Submit your change by clicking on `Create pull request` button.
9. Be patient while Ansibot, our automated script, adds labels, pings the docs maintainers, and kicks off a CI testing run.
10. Keep an eye on your PR - the docs team may ask you for changes.

Reviewing or solving open issues

Review or solve open documentation issues for:

- Ansible projects
- Ansible collections

Reviewing open PRs

Review open documentation pull requests for:

- Ansible projects
- Ansible collections

To add a helpful review, please:

- Test the change if applicable.
- Think if it can be made better (including wording, structure, fixing typos and so on).
- Suggest improvements.
- Approve the change with the `looks good to me` comment.

Opening a new issue and/or PR

If the problem you have noticed is too complex to fix with the `Edit on GitHub` option, and no open issue or PR already documents the problem, please open an issue and/or a PR on the correct underlying repo - `ansible/ansible` for most pages that are not plugin or module documentation. If the documentation page has no `Edit on GitHub` option, check if the page is for a module within a collection. If so, follow the link to the collection on Galaxy and select the `repo` button in the upper right corner to find the source repository for that collection and module. The Collection README file should contain information on how to contribute to that collection, or report issues.

A great documentation GitHub issue or PR includes:

- a specific title
- a detailed description of the problem (even for a PR - it's hard to evaluate a suggested change unless we know what problem it's meant to solve)
- links to other information (related issues/PRs, external documentation, pages on docs.ansible.com, and so on)

Verifying your documentation PR

If you make multiple changes to the documentation on `ansible/ansible`, or add more than a line to it, before you open a pull request, please:

1. Check that your text follows our *Ansible documentation style guide*.
2. Test your changes for rST errors.
3. Build the page, and preferably the entire documentation site, locally.

Note: The following sections apply to documentation sourced from the `ansible/ansible` repo and does not apply to documentation from an individual collection. See the collection README file for details on how to contribute to that collection.

Setting up your environment to build documentation locally

To build documentation locally, ensure you have a working *development environment*.

To work with documentation on your local machine, you need to have python-3.9 or greater and install the Ansible dependencies and documentation dependencies, which are listed in two files to make installation easier:

Drop the `--user` option in the following commands if you use a virtual environment (venv/virtenv).

1. Upgrade pip before installing dependencies (recommended).

   ```
   pip install --user --upgrade pip
   ```

2. Install Ansible dependencies.

   ```
   pip install --user -r requirements.txt
   ```

3. Install either the unpinned or tested documentation dependencies.

   ```
   pip install --user -r docs/docsite/requirements.txt # This file installs unpinned␣
   ↪versions that can cause problems with the Ansible docs build.
   pip install --user -r test/sanity/code-smell/docs-build.requirements.txt # This␣
   ↪file installs tested dependency versions that are used by CI.
   ```

Note: You may need to install these general pre-requisites separately on some systems: - gcc - libyaml - make - pyparsing - wheel - six On macOS with Xcode, you may need to install `six` and `pyparsing` with `--ignore-installed` to get versions that work with `sphinx`.

Note: After checking out `ansible/ansible`, make sure the `docs/docsite/rst` directory has strict enough permissions. It should only be writable by the owner's account. If your default `umask` is not 022, you can use `chmod go-w docs/docsite/rst` to set the permissions correctly in your new branch. Optionally, you can set your `umask` to 022 to make all newly created files on your system (including those created by `git clone`) have the correct permissions.

Testing the documentation locally

To test an individual file for rST errors:

```
rstcheck changed_file.rst
```

Building the documentation locally

Building the documentation is the best way to check for errors and review your changes. Once *rstcheck* runs with no errors, navigate to `ansible/docs/docsite` and then build the page(s) you want to review.

> **Note:** If building on macOS with Python 3.8 or later, you must use Sphinx >= 2.2.2. See #6803 for details.

Building a single rST page

To build a single rST file with the make utility:

```
make htmlsingle rst=path/to/your_file.rst
```

For example:

```
make htmlsingle rst=community/documentation_contributions.rst
```

This process compiles all the links but provides minimal log output. If you're writing a new page or want more detailed log output, refer to the instructions on *Building rST files with sphinx-build*

Note: `make htmlsingle` adds `rst/` to the beginning of the path you provide in `rst=`, so you can't type the filename with autocomplete. Here are the error messages you will see if you get this wrong:

- If you run `make htmlsingle` from the `docs/docsite/rst/` directory: `make: *** No rule to make target `htmlsingle'. Stop.`
- If you run `make htmlsingle` from the `docs/docsite/` directory with the full path to your rST document: `sphinx-build: error: cannot find files ['rst/rst/community/documentation_contributions.rst']`.

Building all the rST pages

To build all the rST files without any module documentation:

```
MODULES=none make webdocs
```

Building module docs and rST pages

To build documentation for a few modules included in `ansible/ansible` plus all the rST files, use a comma-separated list:

```
MODULES=one_module,another_module make webdocs
```

To build all the module documentation plus all the rST files:

```
make webdocs
```

Building rST files with `sphinx-build`

Advanced users can build one or more rST files with the sphinx utility directly. `sphinx-build` returns misleading `undefined label` warnings if you only build a single page, because it does not create internal links. However, `sphinx-build` returns more extensive syntax feedback, including warnings about indentation errors and `x-string without end-string` warnings. This can be useful, especially if you're creating a new page from scratch. To build a page or pages with `sphinx-build`:

```
sphinx-build [options] sourcedir outdir [filenames...]
```

You can specify filenames, or `-a` for all files, or omit both to compile only new/changed files.

For example:

```
sphinx-build -b html -c rst/ rst/dev_guide/ _build/html/dev_guide/ rst/dev_guide/
↪developing_modules_documenting.rst
```

Running the final tests

When you submit a documentation pull request, automated tests are run. Those same tests can be run locally. To do so, navigate to the repository's top directory and run:

```
make clean &&
bin/ansible-test sanity --test docs-build &&
bin/ansible-test sanity --test rstcheck
```

Unfortunately, leftover rST-files from previous document-generating can occasionally confuse these tests. It is therefore safest to run them on a clean copy of the repository, which is the purpose of `make clean`. If you type these three lines one at a time and manually check the success of each, you do not need the `&&`.

Joining the documentation working group

The Documentation Working Group (DaWGs) meets weekly on Tuesdays in the Docs chat (using Matrix or using IRC at irc.libera.chat). For more information, including links to our agenda and a calendar invite, please visit the working group page in the community repo.

See also:

More about testing module documentation

More about documenting modules

1.13.11 Other Tools and Programs

The Ansible community uses a range of tools for working with the Ansible project. This is a list of some of the most popular of these tools.

If you know of any other tools that should be added, this list can be updated by clicking “Edit on GitHub” on the top right of this page.

Popular editors

Emacs

A free, open-source text editor and IDE that supports auto-indentation, syntax highlighting and built in terminal shell(among other things).

- yaml-mode - YAML highlighting and syntax checking.
- jinja2-mode - Jinja2 highlighting and syntax checking.
- magit-mode - Git porcelain within Emacs.
- lsp-mode - Ansible syntax highlighting, auto-completion and diagnostics.

PyCharm

A full IDE (integrated development environment) for Python software development. It ships with everything you need to write python scripts and complete software, including support for YAML syntax highlighting. It's a little overkill for writing roles/playbooks, but it can be a very useful tool if you write modules and submit code for Ansible. Can be used to debug `ansible-core`. For more information, see PyCharm

Sublime

A closed-source, subscription GUI text editor. You can customize the GUI with themes and install packages for language highlighting and other refinements. You can install Sublime on Linux, macOS and Windows. Useful Sublime plugins include:

- GitGutter - shows information about files in a git repository.
- SideBarEnhancements - provides enhancements to the operations on Sidebar of Files and Folders.
- Sublime Linter - a code-linting framework for Sublime Text 3.
- Pretty YAML - prettifies YAML for Sublime Text 2 and 3.
- Yamllint - a Sublime wrapper around yamllint.

vim

An open-source, free command-line text editor. Useful vim plugins include:

- Ansible vim - vim syntax plugin for Ansible 2.x, it supports YAML playbooks, Jinja2 templates, and Ansible's hosts files.
- Ansible vim and neovim plugin - vim plugin (lsp client) for Ansible, it supports autocompletion, syntax highlighting, hover, diagnostics, and goto support.

Visual studio code

An open-source, free GUI text editor created and maintained by Microsoft. Useful Visual Studio Code plugins include:

- Ansible extension by Red Hat - provides autocompletion, syntax highlighting, hover, diagnostics, goto support, and command to run ansible-playbook and ansible-navigator tool for both local and execution-environment setups.
- YAML Support by Red Hat - provides YAML support through yaml-language-server with built-in Kubernetes and Kedge syntax support.

Note: the Visual Studio Code Ansible extension is maintained by the Ansible community and Red Hat.

Development tools

Finding related issues and PRs

There are various ways to find existing issues and pull requests (PRs)

- jctanner's Ansible Tools - miscellaneous collection of useful helper scripts for Ansible development.

Tools for validating playbooks

- Ansible Lint - a highly configurable linter for Ansible playbooks.
- Ansible Review - an extension of Ansible Lint designed for code review.
- Molecule - a testing framework for Ansible plays and roles.
- yamllint - a command-line utility to check syntax validity including key repetition and indentation issues.

Other tools

- Ansible Inventory Grapher - visually displays inventory inheritance hierarchies and at what level a variable is defined in inventory.
- Ansible Shell - an interactive shell for Ansible with built-in tab completion for all the modules.
- Ansible Silo - a self-contained Ansible environment by Docker.
- Ansigenome - a command line tool designed to help you manage your Ansible roles.
- antsibull-changelog - a changelog generator for Ansible collections.
- antsibull-docs - generates docsites for collections and can validate collection documentation.
- ARA - ARA Records Ansible playbooks and makes them easier to understand and troubleshoot with a reporting API, UI and CLI.
- Awesome Ansible - a collaboratively curated list of awesome Ansible resources.
- nanvault - a standalone tool to encrypt and decrypt files in the Ansible Vault format, featuring UNIX-style composability.
- OpsTools-ansible - uses Ansible to configure an environment that provides the support of OpsTools, namely centralized logging and analysis, availability monitoring, and performance monitoring.

If you have a specific Ansible interest or expertise (for example, VMware, Linode, and so on, consider joining a *working group*.

1.13.12 Working with the Ansible collection repositories

- How can I find *editors, linters, and other tools* that will support my Ansible development efforts?
- Where can I find guidance on *coding in Ansible*?
- How do I *create a collection*?
- How do I *rebase my PR*?
- How do I learn about Ansible's *testing (CI) process*?
- How do I *deprecate a module*?

- See Collection developer tutorials for a quick introduction on how to develop and test your collection contributions.

1.14 ansible-core Contributors Guide

1.14.1 Reporting bugs and requesting features

Reporting a bug

Security bugs

Ansible practices responsible disclosure. To report security-related bugs, send an email to security@ansible.com for an immediate response. Do not submit a ticket or post to any public groups.

Bugs in ansible-core

Before reporting a bug, search in GitHub for already reported issues and open pull requests to see if someone has already addressed your issue. Unsure if you found a bug? Report the behavior on the *mailing list or community chat first*.

Also, use the mailing list or chat to discuss whether the problem is in `ansible-core` or a collection, and for "how do I do this" type questions.

You need a free GitHub account to report bugs that affect:

- multiple plugins
- a plugin that remained in the ansible/ansible repo
- the overall functioning of Ansible

How to write a good bug report

If you find a bug, open an issue using the issue template.

Fill out the issue template as completely and as accurately as possible. Include:

- your Ansible version
- the expected behavior and what you've tried, including the exact commands you were using or tasks you are running.
- the current behavior and why you think it is a bug

- the steps to reproduce the bug
- a minimal reproducible example and comments describing examples
- any relevant configurations and the components you used
- any relevant output plus `ansible -vvvv` (debugging) output
- add the output of `ansible-test-env --show` when filing bug reports involving `ansible-test`.

When sharing YAML in playbooks, ensure that you preserve formatting using code blocks. For multiple-file content, use gist.github.com, more durable than Pastebin content.

Requesting a feature

Before you request a feature, check what is *planned for future Ansible Releases*. Check existing pull requests tagged with feature.

To get your feature into Ansible, *submit a pull request*, either against ansible-core or a collection. See also *Requirements to merge your PR*. For `ansible-core`, you can also open an issue in ansible/ansible or in a corresponding collection repository (To find the correct issue tracker, refer to *Bugs in collections*).

1.14.2 The Ansible Development Cycle

Ansible developers (including community contributors) add new features, fix bugs, and update code in many different repositories. The ansible/ansible repository contains the code for basic features and functions, such as copying module code to managed nodes. This code is also known as `ansible-core`. Other repositories contain plugins and modules that enable Ansible to execute specific tasks, like adding a user to a particular database or configuring a particular network device. These repositories contain the source code for collections.

Development on `ansible-core` occurs on two levels. At the macro level, the `ansible-core` developers and maintainers plan releases and track progress with roadmaps and projects. At the micro level, each PR has its own lifecycle.

Development on collections also occurs at the macro and micro levels. Each collection has its own macro development cycle. For more information on the collections development cycle, see *Contributing to Ansible-maintained Collections*. The micro-level lifecycle of a PR is similar in collections and in `ansible-core`.

Macro development: `ansible-core` roadmaps, releases, and projects

If you want to follow the conversation about what features will be added to `ansible-core` for upcoming releases and what bugs are being fixed, you can watch these resources:

- the *Ansible Roadmap*
- the *Ansible Release Schedule*
- the ansible-core project branches and tags
- various GitHub projects - for example:
  - the 2.15 release project
  - the core documentation project

Micro development: the lifecycle of a PR

If you want to contribute a feature or fix a bug in `ansible-core` or in a collection, you must open a **pull request** ("PR" for short). GitHub provides a great overview of how the pull request process works in general. The ultimate goal of any pull request is to get merged and become part of a collection or `ansible-core`. Here's an overview of the PR lifecycle:

- Contributor opens a PR (always against the `devel` branch)
- Ansibot reviews the PR
- Ansibot assigns labels
- Ansibot pings maintainers
- Azure Pipelines runs the test suite
- Developers, maintainers, community review the PR
- Contributor addresses any feedback from reviewers
- Developers, maintainers, community re-review
- PR merged or closed
- PR *backported* to one or more `stable-X.Y` branches (optional, bugfixes only)

Automated PR review: ansibullbot

Because Ansible receives many pull requests, and because we love automating things, we have automated several steps of the process of reviewing and merging pull requests with a tool called Ansibullbot, or Ansibot for short.

Ansibullbot serves many functions:

- Responds quickly to PR submitters to thank them for submitting their PR
- Identifies the community maintainer responsible for reviewing PRs for any files affected
- Tracks the current status of PRs
- Pings responsible parties to remind them of any PR actions for which they may be responsible
- Provides maintainers with the ability to move PRs through the workflow
- Identifies PRs abandoned by their submitters so that we can close them
- Identifies modules abandoned by their maintainers so that we can find new maintainers

Ansibot workflow

Ansibullbot runs continuously. You can generally expect to see changes to your issue or pull request within thirty minutes. Ansibullbot examines every open pull request in the repositories, and enforces state roughly according to the following workflow:

- If a pull request has no workflow labels, it's considered **new**. Files in the pull request are identified, and the maintainers of those files are pinged by the bot, along with instructions on how to review the pull request. (Note: sometimes we strip labels from a pull request to "reboot" this process.)
- If the module maintainer is not `$team_ansible`, the pull request then goes into the **community_review** state.
- If the module maintainer is `$team_ansible`, the pull request then goes into the **core_review** state (and probably sits for a while).
- If the pull request is in **community_review** and has received comments from the maintainer:
  - If the maintainer says `shipit`, the pull request is labeled **shipit**, whereupon the Core team assesses it for final merge.
  - If the maintainer says `needs_info`, the pull request is labeled **needs_info** and the submitter is asked for more info.
  - If the maintainer says **needs_revision**, the pull request is labeled **needs_revision** and the submitter is asked to fix some things.
- If the submitter says `ready_for_review`, the pull request is put back into **community_review** or **core_review** and the maintainer is notified that the pull request is ready to be reviewed again.
- If the pull request is labeled **needs_revision** or **needs_info** and the submitter has not responded lately:
  - The submitter is first politely pinged after two weeks, pinged again after two more weeks and labeled **pending action**, and the issue or pull request will be closed two weeks after that.
  - If the submitter responds at all, the clock is reset.
- If the pull request is labeled **community_review** and the reviewer has not responded lately:
  - The reviewer is first politely pinged after two weeks, pinged again after two more weeks and labeled **pending_action**, and then may be reassigned to `$team_ansible` or labeled **core_review**, or often the submitter of the pull request is asked to step up as a maintainer.

- If Azure Pipelines tests fail, or if the code is not able to be merged, the pull request is automatically put into **needs_revision** along with a message to the submitter explaining why.

There are corner cases and frequent refinements, but this is the workflow in general.

PR labels

There are two types of PR Labels generally: **workflow** labels and **information** labels.

Workflow labels

- **community_review**: Pull requests for modules that are currently awaiting review by their maintainers in the Ansible community.
- **core_review**: Pull requests for modules that are currently awaiting review by their maintainers on the Ansible Core team.
- **needs_info**: Waiting on info from the submitter.
- **needs_rebase**: Waiting on the submitter to rebase.
- **needs_revision**: Waiting on the submitter to make changes.
- **shipit**: Waiting for final review by the core team for potential merge.

Information labels

- **backport**: this is applied automatically if the PR is requested against any branch that is not devel. The bot immediately assigns the labels backport and `core_review`.
- **bugfix_pull_request**: applied by the bot based on the templatized description of the PR.
- **cloud**: applied by the bot based on the paths of the modified files.
- **docs_pull_request**: applied by the bot based on the templatized description of the PR.
- **easyfix**: applied manually, inconsistently used but sometimes useful.
- **feature_pull_request**: applied by the bot based on the templatized description of the PR.
- **networking**: applied by the bot based on the paths of the modified files.
- **owner_pr**: largely deprecated. Formerly workflow, now informational. Originally, PRs submitted by the maintainer would automatically go to **shipit** based on this label. If the submitter is also a maintainer, we notify the other maintainers and still require one of the maintainers (including the submitter) to give a **shipit**.
- **pending_action**: applied by the bot to PRs that are not moving. Reviewed every couple of weeks by the community team, who tries to figure out the appropriate action (closure, asking for new maintainers, and so on).

Special Labels

- **new_plugin**: this is for new modules or plugins that are not yet in Ansible.

Note: *new_plugin* kicks off a completely separate process, and frankly it doesn't work very well at present. We're working our best to improve this process.

Human PR review

After Ansibot reviews the PR and applies labels, the PR is ready for human review. The most likely reviewers for any PR are the maintainers for the module that PR modifies.

Each module has at least one assigned *maintainer*, listed in the BOTMETA.yml file.

The maintainer's job is to review PRs that affect that module and decide whether they should be merged (`shipit`) or revised (`needs_revision`). We'd like to have at least one community maintainer for every module. If a module has no community maintainers assigned, the maintainer is listed as `$team_ansible`.

Once a human applies the `shipit` label, the *committers* decide whether the PR is ready to be merged. Not every PR that gets the `shipit` label is actually ready to be merged, but the better our reviewers are, and the better our guidelines are, the more likely it will be that a PR that reaches **shipit** will be mergeable.

Making your PR merge-worthy

We do not merge every PR. Here are some tips for making your PR useful, attractive, and merge-worthy.

Creating changelog fragments

Changelogs help users and developers keep up with changes to ansible-core and Ansible collections. Ansible and many collections build changelogs for each release from fragments. For ansible-core and collections using this model, you **must** add a changelog fragment to any PR that changes functionality or fixes a bug.

You do not need a changelog fragment for PRs that:

- add new modules and plugins, because Ansible tooling does that automatically;
- contain only documentation changes.

Note: Some collections require a changelog fragment for every pull request. They use the `trivial:` section for entries mentioned above that will be skipped when building a release changelog.

More precisely:

- Every bugfix PR must have a changelog fragment. The only exception are fixes to a change that has not yet been included in a release.
- Every feature PR must have a changelog fragment.
- New modules and plugins (including jinja2 filter and test plugins) must have `version_added` entries set correctly in their documentation, and do not need a changelog fragment. The tooling detects new modules and plugins by their `version_added` values and announces them in the next release's changelog automatically.

We build short summary changelogs for minor releases as well as for major releases. If you backport a bugfix, include a changelog fragment with the backport PR.

Creating a changelog fragment

A basic changelog fragment is a `.yaml` or `.yml` file placed in the `changelogs/fragments/` directory. Each file contains a yaml dict with keys like `bugfixes` or `major_changes` followed by a list of changelog entries of bugfixes or features. Each changelog entry is rst embedded inside of the yaml file which means that certain constructs would need to be escaped so they can be interpreted by rst and not by yaml (or escaped for both yaml and rst if you prefer). Each PR **must** use a new fragment file rather than adding to an existing one, so we can trace the change back to the PR that introduced it.

PRs which add a new module or plugin do not necessarily need a changelog fragment. See the previous section *Creating changelog fragments*. Also see the next section *Changelog fragment entry format* for the precise format changelog fragments should have.

To create a changelog entry, create a new file with a unique name in the `changelogs/fragments/` directory of the corresponding repository. The file name should include the PR number and a description of the change. It must end with the file extension `.yaml` or `.yml`. For example: `40696-user-backup-shadow-file.yaml`

A single changelog fragment may contain multiple sections but most will only contain one section. The toplevel keys (bugfixes, major_changes, and so on) are defined in the config file for our release note tool. Here are the valid sections and a description of each:

breaking_changes
MUST include changes that break existing playbooks or roles. This includes any change to existing behavior that forces users to update tasks. Breaking changes means the user MUST make a change when they update. Breaking changes MUST only happen in a major release of the collection. Write in present tense and clearly describe the new behavior that the end user must now follow. Displayed in both the changelogs and the *Porting Guides*.

```yaml
breaking_changes:
  - ansible-test - automatic installation of requirements for cloud test plugins no longer occurs. The affected test plugins are ``aws``, ``azure``, ``cs``, ``hcloud``, ``nios``, ``opennebula``, ``openshift`` and ``vcenter``. Collections should instead use one of the supported integration test requirements files, such as the ``tests/integration/requirements.txt`` file (https://github.com/ansible/ansible/pull/75605).
```

major_changes
Major changes to ansible-core or a collection. SHOULD NOT include individual module or plugin changes. MUST include non-breaking changes that impact all or most of a collection (for example, updates to support a new SDK version across the collection). Major changes mean the user can CHOOSE to make a change when they update but do not have to. Could be used to announce an important upcoming EOL or breaking change in a future release. (ideally 6 months in advance, if known. See this example). Write in present tense and describe what is new. Optionally, include a 'Previously…" sentence to help the user identify where old behavior should now change. Displayed in both the changelogs and the *Porting Guides*.

```yaml
major_changes:
  - ansible-test - all cloud plugins which use containers can now be used with all POSIX and Windows hosts. Previously the plugins did not work with Windows at all, and support for hosts created with the ``--remote`` option was inconsistent (https://github.com/ansible/ansible/pull/74216).
```

minor_changes
Minor changes to ansible-core, modules, or plugins. This includes new parameters added to modules, or non-breaking behavior changes to existing parameters, such as adding additional values to choices[]. Minor changes are enhancements, not bug fixes. Write in present tense.

```
minor_changes:
  - lineinfile - add warning when using an empty regexp (https://github.com/ansible/
↪ansible/issues/29443).
```

deprecated_features

Features that have been deprecated and are scheduled for removal in a future release. Use past tense and include an alternative, where available for what is being deprecated.. Displayed in both the changelogs and the *Porting Guides*.

```
deprecated_features:
  - include action - is deprecated in favor of ``include_tasks``, ``import_tasks``␣
↪and ``import_playbook`` (https://github.com/ansible/ansible/pull/71262).
```

removed_features

Features that were previously deprecated and are now removed. Use past tense and include an alternative, where available for what is being deprecated. Displayed in both the changelogs and the *Porting Guides*.

```
removed_features:
  - _get_item() alias - removed from callback plugin base class which had been␣
↪deprecated in favor of ``_get_item_label()`` (https://github.com/ansible/ansible/
↪pull/70233).
```

security_fixes

Fixes that address CVEs or resolve security concerns. MUST use security_fixes for any CVEs. Use present tense. Include links to CVE information.

```
security_fixes:
  - set_options -do not include params in exception when a call to ``set_options``␣
↪fails. Additionally, block the exception that is returned from being displayed to␣
↪stdout. (CVE-2021-3620).
```

bugfixes

Fixes that resolve issues. SHOULD not be used for minor enhancements (use `minor_change` instead). Use past tense to describe the problem and present tense to describe the fix.

```
bugfixes:
  - ansible_play_batch - variable included unreachable hosts. Fix now saves␣
↪unreachable hosts between plays by adding them to the PlayIterator's ``_play._
↪removed_hosts`` (https://github.com/ansible/ansible/issues/66945).
```

known_issues

Known issues that are currently not fixed or will not be fixed. Use present tense and where available, use imperative tense for a workaround.

```
known_issues:
  - ansible-test - tab completion anywhere other than the end of the command with␣
↪the new composite options provides incorrect results (https://github.com/kislyuk/
↪argcomplete/issues/351).
```

Each changelog entry must contain a link to its issue between parentheses at the end. If there is no corresponding issue, the entry must contain a link to the PR itself.

Most changelog entries are `bugfixes` or `minor_changes`. The changelog tool also supports `trivial`, which are not listed in the actual changelog output but are used by collections repositories that require a changelog fragment for each PR.

Changelog fragment entry format

When writing a changelog entry, use the following format:

```
- scope - description starting with a lowercase letter and ending with a period at the␣
↪very end. Multiple sentences are allowed (https://github.com/reference/to/an/issue or,␣
↪if there is no issue, reference to a pull request itself).
```

The scope is usually a module or plugin name or group of modules or plugins, for example, `lookup plugins`. While module names can (and should) be mentioned directly (`foo_module`), plugin names should always be followed by the type (`foo inventory plugin`).

For changes that are not really scoped (for example, which affect a whole collection), use the following format:

```
- Description starting with an uppercase letter and ending with a dot at the very end.␣
↪Multiple sentences are allowed (https://github.com/reference/to/an/issue or, if there␣
↪is no issue, reference to a pull request itself).
```

Here are some examples:

```
bugfixes:
  - apt_repository - fix crash caused by ``cache.update()`` raising an ``IOError``
    due to a timeout in ``apt update`` (https://github.com/ansible/ansible/issues/51995).
```

```
minor_changes:
  - lineinfile - add warning when using an empty regexp (https://github.com/ansible/
↪ansible/issues/29443).
```

```
bugfixes:
  - copy - the module was attempting to change the mode of files for
    remote_src=True even if mode was not set as a parameter.  This failed on
    filesystems which do not have permission bits (https://github.com/ansible/ansible/
↪issues/29444).
```

You can find more example changelog fragments in the changelog directory for the 2.12 release.

After you have written the changelog fragment for your PR, commit the file and include it with the pull request.

Changelog fragment entry format for new playbooks

While new modules, plugins, and roles are mentioned automatically in the generated changelog, playbooks are not. To make sure they are mentioned, a changelog fragment in a specific format is needed:

```
# A new playbook:
add object.playbook:
  - # This should be the short (non-FQCN) name of the playbook.
    name: wipe_server
    # The description should be in the same format as short_description for
    # plugins and modules: it should start with an upper-case letter and
    # not have a period at the end.
    description: Wipes a server
```

Backporting merged PRs in `ansible-core`

All `ansible-core` PRs must be merged to the `devel` branch first. After a pull request has been accepted and merged to the `devel` branch, the following instructions will help you create a pull request to backport the change to a previous stable branch.

We do **not** backport features.

Note: These instructions assume that:

- `stable-2.14` is the targeted release branch for the backport
- `https://github.com/ansible/ansible.git` is configured as a `git remote` named `upstream`. If you do not use a `git remote` named `upstream`, adjust the instructions accordingly.
- `https://github.com/<yourgithubaccount>/ansible.git` is configured as a `git remote` named `origin`. If you do not use a `git remote` named `origin`, adjust the instructions accordingly.

1. Prepare your devel, stable, and feature branches:

```
git fetch upstream
git checkout -b backport/2.14/[PR_NUMBER_FROM_DEVEL] upstream/stable-2.14
```

1. Cherry pick the relevant commit SHA from the devel branch into your feature branch, handling merge conflicts as necessary:

```
git cherry-pick -x [SHA_FROM_DEVEL]
```

1. Add a *changelog fragment* for the change, and commit it.
2. Push your feature branch to your fork on GitHub:

```
git push origin backport/2.14/[PR_NUMBER_FROM_DEVEL]
```

1. Submit the pull request for `backport/2.14/[PR_NUMBER_FROM_DEVEL]` against the `stable-2.14` branch
2. The Release Manager will decide whether to merge the backport PR before the next minor release. There isn't any need to follow up. Just ensure that the automated tests (CI) are green.

Note: The branch name `backport/2.14/[PR_NUMBER_FROM_DEVEL]` is somewhat arbitrary but conveys meaning about the purpose of the branch. This branch name format is not required, but it can be helpful, especially when making multiple backport PRs for multiple stable branches.

Note: If you prefer, you can use CPython's cherry-picker tool (`pip install --user 'cherry-picker >= 1.3.2'`) to backport commits from devel to stable branches in Ansible. Take a look at the cherry-picker documentation for details on installing, configuring, and using it.

If you have a specific Ansible interest or expertise (for example, VMware, Linode, and so on, consider joining a *working group*.

1.14.3 Working with the Ansible repo

- I want to make my first code changes to a collection or to `ansible-core`. How do I *set up my Python development environment*?
- I would like to get more efficient as a developer. How can I find *editors, linters, and other tools* that will support my Ansible development efforts?
- I want my code to meet Ansible's guidelines. Where can I find guidance on *coding in Ansible*?
- I would like to connect Ansible to a new API or other resource. How do I *create a collection*?
- My pull request is marked `needs_rebase`. How do I *rebase my PR*?
- I am using an older version of Ansible and want a bug fixed in my version that has already been fixed on the `devel` branch. How do I *backport a bugfix PR*?
- I have an open pull request with a failing test. How do I learn about Ansible's *testing (CI) process*?
- I am ready to step up as a collection maintainer. What are the *guidelines for maintainers*?
- A module in a collection I maintain is obsolete. How do I *deprecate a module*?

1.15 Advanced Contributor Guide

This guide focuses on contributors who are committers, GitHub admins, or release managers.

1.15.1 Committers Guidelines

These are the guidelines for people with commit privileges on the repositories in the ansible and ansible-collections GitHub organizations.

Committers of Ansible-core are necessarily Red Hat employees acting as members of the Ansible Core team. Committers of Ansible collections are members of the community or Ansible Engineering. Please read the guidelines before you commit.

These guidelines apply to everyone. At the same time, this is NOT a process document. So just use good judgment. You have been given commit access because we trust your judgment.

That said, use the trust wisely.

If you abuse the trust and break components and builds, and so on, the trust level falls and you may be asked not to commit or you may lose your commit privileges.

Features, high-level design, and roadmap of ansible-core

As a core team member, you are an integral part of the team that develops the *roadmap*. Please be engaged, and push for the features and fixes that you want to see. Also keep in mind that Red Hat, as a company, will commit to certain features, fixes, APIs, and so on, for various releases. Red Hat, the company, and the Ansible team must get these changes completed and released as scheduled. Obligations to users, the community, and customers must come first. Because of these commitments, a feature you want to develop yourself may not get into a release if it affects a lot of other parts within Ansible.

Any other new features and changes to high level design should go through the proposal process (TBD), to ensure the community and core team have had a chance to review the idea and approve it. The core team has sole responsibility for merging new features based on proposals to Ansible-core.

Features, high-level design, and roadmap of Ansible collections

Collections maintainers define features, high-level design, and roadmap of the collections themselves and are responsible for merging new features to Ansible collections based on proposals discussed with their communities.

Our workflow on GitHub

As a committer, you may already know this, but our workflow forms a lot of our team policies. Please ensure you are aware of the following workflow steps:

- Fork the repository upon which you want to do some work to your own personal repository
- Work on the specific branch upon which you need to commit
- Create a pull request back to the upstream repository and tag the people you would like to review; assign someone as the primary "owner" of your pull request
- Adjust code as necessary based on the comments provided
- Ask someone from the repository committers to do a final review and merge

Addendum to workflow for committers:

The Core Team is aware that this can be a difficult process at times. Sometimes, the team breaks the rules by making direct commits or merging their own pull requests. This section is a set of guidelines. If you are changing a comma in documentation, or making a very minor change, you can use your best judgement. This is another trust thing. The process is critical for any major change, but for little things or getting something done quickly, use your best judgement and make sure people on the team are aware of your work.

Roles on Core

- Core committers: Fine to do pull requests for most things, but we should have a timebox. Hanging pull requests may merge on the judgement of these developers.
- *Module maintainers*: Module maintainers own specific modules and have indirect commit access through the current module pull request mechanisms.
- *Collection maintainers*: Collection maintainers own specific collections and have commit access to them. Each collection can set its own rules for contributions.

General rules

Individuals with direct commit access are entrusted with powers that allow them to do a broad variety of things–probably more than we can write down. Rather than rules, treat these as general *guidelines*, individuals with this power are expected to use their best judgement.

- Do NOT
  - Commit directly.
  - Merge your own pull requests. Someone else should have a chance to review and approve the pull request merge. If you are a Core Committer, you have a small amount of leeway here for very minor changes.
  - Forget about alternate environments. Consider the alternatives–yes, people have bad environments, but they are the ones who need us the most.

  - Drag your community team members down. Discuss the technical merits of any pull requests you review. Avoid negativity and personal comments. For more guidance on being a good community member, read our *Community Code of Conduct*.
  - Forget about the maintenance burden. High-maintenance features may not be worth adding.
  - Break playbooks. Always keep backwards compatibility in mind.
  - Forget to keep it simple. Complexity breeds all kinds of problems.

- Do
  - Squash, avoid merges whenever possible, use GitHub's squash commits or cherry pick if needed (bisect thanks you).
  - Be active. Committers who have no activity on the project (through merges, triage, commits, and so on) will have their permissions suspended.
  - Consider backwards compatibility (goes back to "do not break existing playbooks").
  - Write *tests* and be sure that other's pull requests you are reviewing are covered well. Pull requests with tests are looked at with more priority than pull requests without tests that should have them included. While not all changes require tests, be sure to add them for new features, bug fixes, and functionality changes.
  - Discuss with other committers, specially when you are unsure of something.
  - Document! If your pull request is a new feature or a change to behavior, make sure you have updated all associated documentation or have notified the right people to do so. It also helps to add the version of `ansible-core` or `collection` against which this documentation is compatible (to avoid confusion between stable and devel docs, for backwards compatibility, and so on).
  - Consider scope, sometimes a fix can be generalized.
  - Keep it simple, then things are maintainable, debuggable, and intelligible.

Committers are expected to continue to follow the same community and contribution guidelines followed by the rest of the Ansible community.

1.15.2 Release Manager Guidelines

Topics

The release manager's purpose is to ensure a smooth release. To achieve that goal, they need to coordinate between:

- Developers with commit privileges on the Ansible GitHub repository
- Contributors without commit privileges
- The community
- Ansible documentation team

Pre-releases: what and why

Pre-releases exist to draw testers. They give people who don't feel comfortable running from source control a means to get an early version of the code to test and give us feedback. To ensure we get good feedback about a release, we need to make sure all major changes in a release are put into a pre-release. Testers must be given time to test those changes before the final release. Ideally we want there to be sufficient time between pre-releases for people to install and test one version for a span of time. Then they can spend more time using the new code than installing the latest version.

The right length of time for a tester is probably around two weeks. However, for our three-to-four month development cycle to work, we compress this down to one week; any less runs the risk of people spending more time installing the code instead of running it. However, if there's a time crunch (with a release date that cannot slip), it is better to release with new changes than to hold back those changes to give people time to test between. People cannot test what is not released, so we have to get those tarballs out there even if people feel they have to install more frequently.

Beta releases

In a beta release, we know there are still bugs. We will continue to accept fixes for these. Although we review these fixes, sometimes they can be invasive or potentially destabilize other areas of the code.

During the beta, we will no longer accept feature submissions.

Release candidates

In a release candidate, we've fixed all known blockers. Any remaining bugfixes are ones that we are willing to leave out of the release. At this point we need user testing to determine if there are any other blocker bugs lurking.

Blocker bugs generally are those that cause significant problems for users. Regressions are more likely to be considered blockers because they will break present users' usage of Ansible.

The Release Manager will cherry-pick fixes for new release blockers. The release manager will also choose whether to accept bugfixes for isolated areas of the code or defer those to the next minor release. By themselves, non-blocker bugs will not trigger a new release; they will only make it into the next major release if blocker bugs require that a new release be made.

The last RC should be as close to the final as possible. The following things may be changed:

- Version numbers are changed automatically and will differ as the pre-release tags are removed from the versions.
- Tests and `docs/docsite/` can differ if really needed as they do not break runtime. However, the release manager may still reject them as they have the potential to cause breakage that will be visible during the release process.

Note: We want to specifically emphasize that code (in `bin/`, `lib/ansible/`, and `setup.py`) must be the same unless there are extraordinary extenuating circumstances. If there are extenuating circumstances, the Release Manager is responsible for notifying groups which would want to test the code.

Ansible release process

The release process is kept in a separate document so that it can be easily updated during a release. If you need access to edit this, please ask one of the current release managers to add you.

1.15.3 GitHub Admins

Topics

GitHub Admins have more permissions on GitHub than normal contributors or even committers. There are a few responsibilities that come with that increased power.

Adding and removing committers

The Ansible Team will periodically review who is actively contributing to Ansible to grant or revoke contributors' ability to commit on their own. GitHub Admins are the people who have the power to actually manage the GitHub permissions.

Changing branch permissions for releases

When we make releases we make people go through a *Release Manager Guidelines* to push commits to that branch. The GitHub admins are responsible for setting the branch so only the Release Manager can commit to the branch when the release process reaches that stage and later opening the branch once the release has been made. The Release manager will let the GitHub Admin know when this needs to be done.

See also:

The GitHub Admin Process Docs for instructions on how to change branch permissions.

1.16 Ansible documentation style guide

Welcome to the Ansible style guide! To create clear, concise, consistent, useful materials on docs.ansible.com, follow these guidelines:

1.16.1 Linguistic guidelines

We want the Ansible documentation to be:

- clear
- direct
- conversational
- easy to translate

We want reading the docs to feel like having an experienced, friendly colleague explain how Ansible works.

Stylistic cheat-sheet

This cheat-sheet illustrates a few rules that help achieve the "Ansible tone":

Rule	Good example	Bad example
Use active voice	You can run a task by	A task can be run by
Use the present tense	This command creates a	This command will create a
Address the reader	As you expand your inventory	When the number of managed nodes grows
Use standard English	Return to this page	Hop back to this page
Use American English	The color of the output	The colour of the output

Header case

Headers should be written in sentence case. For example, this section's title is `Header case`, not `Header Case` or `HEADER CASE`.

Avoid using Latin phrases

Latin words and phrases like `e.g.` or `etc.` are easily understood by English speakers. They may be harder to understand for others and are also tricky for automated translation.

Use the following English terms in place of Latin terms or abbreviations:

Latin	English
i.e	in other words
e.g.	for example
etc	and so on
via	by/ through
vs./versus	rather than/against

1.16.2 reStructuredText guidelines

The Ansible documentation is written in reStructuredText and processed by Sphinx. We follow these technical or mechanical guidelines on all rST pages:

Header notation

Section headers in reStructuredText can use a variety of notations. Sphinx will 'learn on the fly' when creating a hierarchy of headers. To make our documents easy to read and to edit, we follow a standard set of header notations. We use:

- ### with overline, for parts:

```
###############
Developer guide
###############
```

- *** with overline, for chapters:

```
*******************
Ansible style guide
*******************
```

- === for sections:

```
Mechanical guidelines
=====================
```

- --- for subsections:

```
Internal navigation
-------------------
```

- ^^^ for sub-subsections:

```
Adding anchors
^^^^^^^^^^^^^^
```

- """ for paragraphs:

```
Paragraph that needs a title
""""""""""""""""""""""""""""
```

Syntax highlighting - Pygments

The Ansible documentation supports a range of Pygments lexers for syntax highlighting to make our code examples look good. Each code-block must be correctly indented and surrounded by blank lines.

The Ansible documentation allows the following values:

- none (no highlighting)
- ansible-output (a custom lexer for Ansible output)
- bash
- console
- csharp
- ini
- json
- powershell
- python
- rst
- sh
- shell
- shell-session
- text
- yaml
- yaml+jinja

For example, you can highlight Python code using following syntax:

```
.. code-block:: python

   def my_beautiful_python_code():
      pass
```

Internal navigation

Anchors (also called labels) and links work together to help users find related content. Local tables of contents also help users navigate quickly to the information they need. All internal links should use the `:ref:` syntax. Every page should have at least one anchor to support internal `:ref:` links. Long pages, or pages with multiple levels of headers, can also include a local TOC.

Note: Avoid raw URLs. RST and sphinx allow :`https://my.example.com`, but this is unhelpful for those using screen readers. `:ref:` links automatically pick up the header from the anchor, but for external links, always use the :`link title <link-url>`_ format.

Adding anchors

- Include at least one anchor on every page
- Place the main anchor above the main header
- If the file has a unique title, use that for the main page anchor:

```
.. _unique_page::
```

- You may also add anchors elsewhere on the page

Adding internal links

- All internal links must use `:ref:` syntax. These links both point to the anchor defined above:

```
:ref:`unique_page`
:ref:`this page <unique_page>`
```

The second example adds custom text for the link.

Adding links to modules and plugins

Ansible 2.10 and later require the extended Fully Qualified Collection Name (FQCN) as part of the links:

```
ansible_collections. + FQCN + _module
```

For example:

```
:ref:`ansible.builtin.first_found lookup plugin <ansible_collections.ansible.
↪builtin.first_found_lookup>`
```

displays as ansible.builtin.first_found lookup plugin.

Modules require different suffixes from other plugins:

- Module links use this extended FQCN module name with `_module` for the anchor.
- Plugin links use this extended FQCN plugin name with the plugin type (`_connection` for example).

```
:ref:`arista.eos.eos_config <ansible_collections.arista.eos.eos_config_module>`
:ref:`kubernetes.core.kubectl connection plugin <ansible_collections.kubernetes.core.
↪kubectl_connection>`
```

Note: `ansible.builtin` is the FQCN for modules included in `ansible.base`. Documentation links are the only place you prepend `ansible_collections` to the FQCN. This is used by the documentation build scripts to correctly fetch documentation from collections on Ansible Galaxy.

Adding local TOCs

The page you're reading includes a local TOC. If you include a local TOC:

- place it below, not above, the main heading and (optionally) introductory text
- use the `:local:` directive so the page's main header is not included
- do not include a title

The syntax is:

```
.. contents::
   :local:
```

1.16.3 Accessibility guidelines

Ansible documentation has a goal to be more accessible. Use the following guidelines to help us reach this goal.

Images and alternative text
Ensure all icons, images, diagrams, and non text elements have a meaningful alternative-text description. Do not include screen captures of CLI output. Use `code-block` instead.

```
.. image:: path/networkdiag.png
   :width: 400
   :alt: SpiffyCorp network diagram
```

Links and hypertext
URLs and cross-reference links have descriptive text that conveys information about the content of the linked target. See *Internal navigation* for how to format links.

Tables
Tables have a simple, logical reading order from left to right, and top to bottom. Tables include a header row and avoid empty or blank table cells. Label tables with a descriptive title.

```
.. table:: File descriptions

  +----------+----------------------------+
  |File      |Purpose                     |
  +==========+============================+
  |foo.txt   |foo configuration settings  |
  +----------+----------------------------+
  |bar.txt   |bar configuration settings  |
  +----------+----------------------------+
```

Colors and other visual information

- Avoid instructions that rely solely on sensory characteristics. For example, do not use `Click the square, blue button to continue.`
- Convey information by methods and not by color alone.
- Ensure there is sufficient contrast between foreground and background text or graphical elements in images and diagrams.
- Instructions for navigating through an interface make sense without directional indicators such as left, right, above, and below.

Accessibility resources

Use the following resources to help test your documentation changes:

- axe DevTools browser extension - Highlights accessibility issues on a website page.
- WAVE browser extension from WebAIM - another accessibility tester.
- Orca screen reader - Common tool used by people with vision impairments.
- color filter - For color-blind testing.

1.16.4 More resources

These pages offer more help with grammatical, stylistic, and technical rules for documentation.

Basic rules

Use standard American English

Ansible uses Standard American English. Watch for common words that are spelled differently in American English (color vs colour, organize vs organise, and so on).

Write for a global audience

Everything you say should be understandable by people of different backgrounds and cultures. Avoid idioms and regionalism and maintain a neutral tone that cannot be misinterpreted. Avoid attempts at humor.

Follow naming conventions

Always follow naming conventions and trademarks.

Use clear sentence structure

Clear sentence structure means:

- Start with the important information first.
- Avoid padding/adding extra words that make the sentence harder to understand.
- Keep it short - Longer sentences are harder to understand.

Some examples of improving sentences:

Bad:
The unwise walking about upon the area near the cliff edge may result in a dangerous fall and therefore it is recommended that one remains a safe distance to maintain personal safety.

Better:
Danger! Stay away from the cliff.

Bad:
Furthermore, large volumes of water are also required for the process of extraction.

Better:
Extraction also requires large volumes of water.

Avoid verbosity

Write short, succinct sentences. Avoid terms like:

- "…as has been said before,"
- "..each and every,"
- "…point in time,"
- "…in order to,"

Highlight menu items and commands

When documenting menus or commands, it helps to **bold** what is important.

For menu procedures, bold the menu names, button names, and so on to help the user find them on the GUI:

1. On the **File** menu, click **Open**.
2. Type a name in the **User Name** field.
3. In the **Open** dialog box, click **Save**.
4. On the toolbar, click the **Open File** icon.

For code or command snippets, use the RST code-block directive:

```
.. code-block:: bash

  ssh my_vyos_user@vyos.example.net
  show config
```

Voice Style

The essence of the Ansible writing style is short sentences that flow naturally together. Mix up sentence structures. Vary sentence subjects. Address the reader directly. Ask a question. And when the reader adjusts to the pace of shorter sentences, write a longer one.

- Write how real people speak…
- …but try to avoid slang and colloquialisms that might not translate well into other languages.
- Say big things with small words.
- Be direct. Tell the reader exactly what you want them to do.
- Be honest.
- Short sentences show confidence.
- Grammar rules are meant to be bent, but only if the reader knows you are doing this.
- Choose words with fewer syllables for faster reading and better understanding.
- Think of copy as one-on-one conversations rather than as a speech. It's more difficult to ignore someone who is speaking to you directly.
- When possible, start task-oriented sentences (those that direct a user to do something) with action words. For example: Find software… Contact support… Install the media…. and so forth.

Active Voice

Use the active voice ("Start Linuxconf by typing…") rather than passive ("Linuxconf can be started by typing…") whenever possible. Active voice makes for more lively, interesting reading. Also avoid future tense (or using the term "will") whenever possible For example, future tense ("The screen will display…") does not read as well as an active voice ("The screen displays"). Remember, the users you are writing for most often refer to the documentation while they are using the system, not after or in advance of using the system.

Trademark Usage

Why is it important to use the TM, SM, and ® for our registered marks?

Before a trademark is registered with the United States Patent and Trademark Office it is appropriate to use the TM or SM symbol depending whether the product is for goods or services. It is important to use the TM or SM as it is notification to the public that Ansible claims rights to the mark even though it has not yet been registered.

Once the trademark is registered, it is appropriate to use the symbol in place of the TM or SM. The symbol designation must be used in conjunction with the trademark if Ansible is to fully protect its rights. If we don't protect these marks, we run the risk of losing them in the way of Aspirin or Trampoline or Escalator.

General Rules:

Trademarks should be used on 1st references on a page or within a section.

Use Red Hat® Ansible® Automation Platform or Ansible®, on first reference when referring to products.

Use "Ansible" alone as the company name, as in "Ansible announced quarterly results," which is not marked.

Also add the trademark disclaimer. * When using Ansible trademarks in the body of written text, you should use the following credit line in a prominent place, usually a footnote.

> For Registered Trademarks: - [Name of Trademark] is a registered trademark of Red Hat, Inc. in the United States and other countries.
>
> For Unregistered Trademarks (TMs/SMs): - [Name of Trademark] is a trademark of Red Hat, Inc. in the United States and other countries.
>
> For registered and unregistered trademarks: - [Name of Trademark] is a registered trademark and [Name of Trademark] is a trademark of Red Hat, Inc. in the United States and other countries.

Guidelines for the proper use of trademarks:

> Always distinguish trademarks from surround text with at least initial capital letters or in all capital letters.

Always use proper trademark form and spelling.

Never use a trademark as a noun. Always use a trademark as an adjective modifying the noun.

> Correct: Red Hat® Ansible® Automation Platform system performance is incredible.
>
> Incorrect: Ansible's performance is incredible.

Never use a trademark as a verb. Trademarks are products or services, never actions.

> Correct: "Orchestrate your entire network using Red Hat® Ansible® Automation Platform."
>
> Incorrect: "Ansible your entire network."

Never modify a trademark to a plural form. Instead, change the generic word from the singular to the plural.

> Correct: "Corporate demand for Red Hat® Ansible® Automation Platform software is surging."
>
> Incorrect: "Corporate demand for Ansible is surging."

Never modify a trademark from its possessive form, or make a trademark possessive. Always use it in the form it has been registered.

Never translate a trademark into another language.

Never use trademarks to coin new words or names.

Never use trademarks to create a play on words.

Never alter a trademark in any way including through unapproved fonts or visual identifiers.

Never abbreviate or use any Ansible trademarks as an acronym.

The importance of Ansible trademarks

The Ansible trademark and the "A" logo in a shaded circle are our most valuable assets. The value of these trademarks encompass the Ansible Brand. Effective trademark use is more than just a name, it defines the level of quality the customer will receive and it ties a product or service to a corporate image. A trademark may serve as the basis for many of our everyday decisions and choices. The Ansible Brand is about how we treat customers and each other. In order to continue to build a stronger more valuable Brand we must use it in a clear and consistent manner.

The mark consists of the letter "A" in a shaded circle. As of 5/11/15, this was a pending trademark (registration in process).

Common Ansible Trademarks

- Ansible®

Other Common Trademarks and Resource Sites:

- Linux is a registered trademark of Linus Torvalds.
- UNIX® is a registered trademark of The Open Group.
- Microsoft, Windows, Vista, XP, and NT are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. https://www.microsoft.com/en-us/legal/intellectualproperty/trademarks/en-us.aspx
- Apple, Mac, Mac OS, Macintosh, Pages and TrueType are either registered trademarks or trademarks of Apple Computer, Inc. in the United States and/or other countries. https://www.apple.com/legal/intellectual-property/trademark/appletmlist.html
- Adobe, Acrobat, GoLive, InDesign, Illustrator, PostScript , PhotoShop and the OpenType logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. https://www.adobe.com/legal/permissions/trademarks.html
- Macromedia and Macromedia Flash are trademarks of Macromedia, Inc. https://www.adobe.com/legal/permissions/trademarks.html
- IBM is a registered trademark of International Business Machines Corporation. https://www.ibm.com/legal/us/en/copytrade.shtml
- Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, Intel Core, Intel Inside, Intel Inside logo, Itanium, Itanium Inside, Pentium, Pentium Inside,VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. https://www.intel.com/content/www/us/en/legal/trademarks.html

Grammar and Punctuation

Common Styles and Usage, and Common Mistakes

Ansible

- Write "Ansible." Not "Ansible, Inc." or "AnsibleWorks The only exceptions to this rule are when we're writing legal or financial statements.
- Never use the logotype by itself in body text. Always keep the same font you are using the rest of the sentence.

- A company is singular in the US. In other words, Ansible is an "it," not a "they."

Capitalization

If it's not a real product, service, or department at Ansible, don't capitalize it. Not even if it seems important. Capitalize only the first letter of the first word in headlines.

Colon

A colon is generally used before a list or series: - The Triangle Area consists of three cities: Raleigh, Durham, and Chapel Hill.

But not if the list is a complement or object of an element in the sentence: - Before going on vacation, be sure to (1) set the alarm, (2) cancel the newspaper, and (3) ask a neighbor to collect your mail.

Use a colon after "as follows" and "the following" if the related list comes immediately after: wedge The steps for changing directories are as follows:

1. Open a terminal.
2. Type cd…

Use a colon to introduce a bullet list (or dash, or icon/symbol of your choice):

> In the Properties dialog box, you'll find the following entries:
>
> - Connection name
> - Count
> - Cost per item

Commas

Use serial commas, the comma before the "and" in a series of three or more items:

- "Item 1, item 2, and item 3."

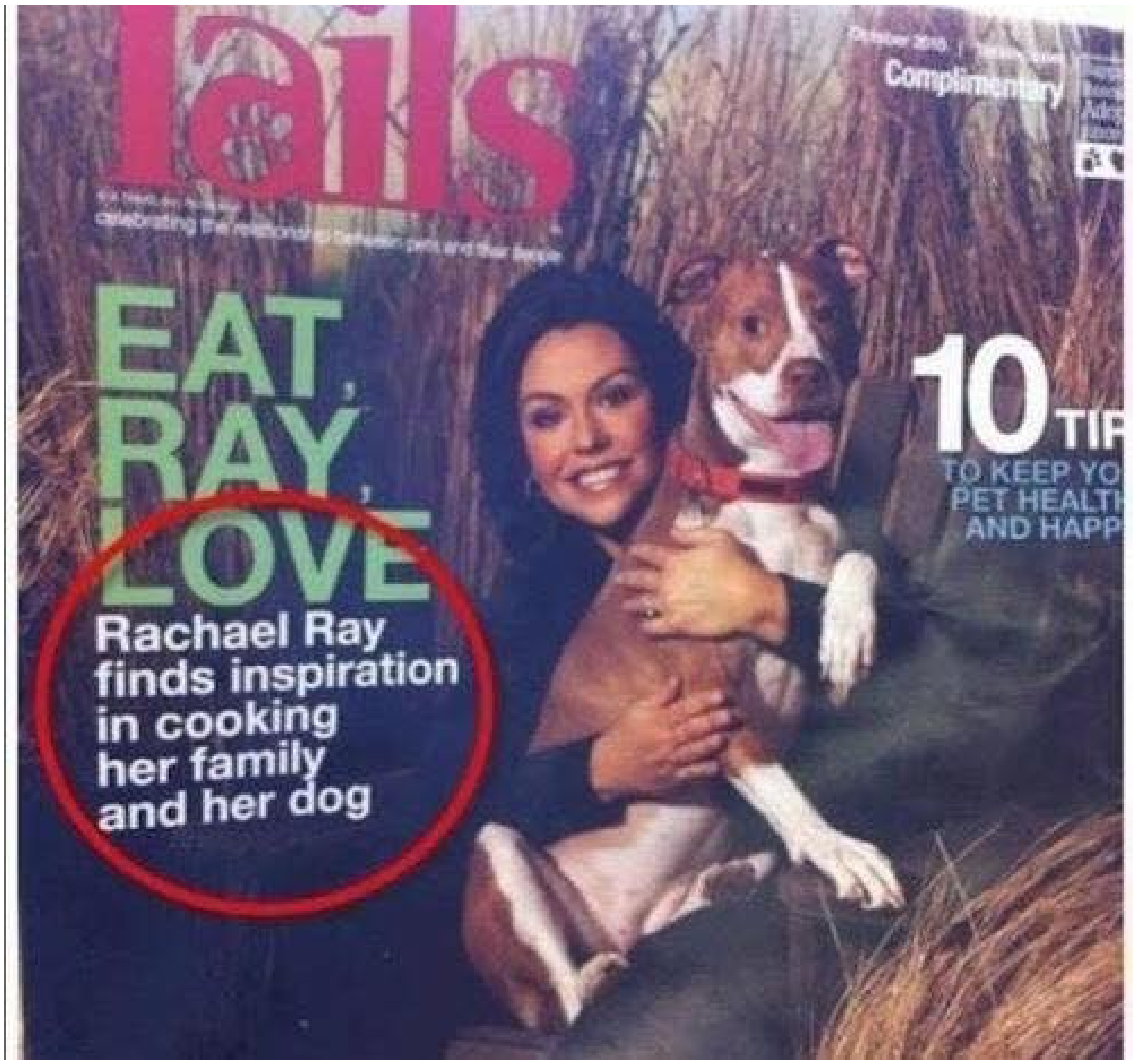
It's easier to read that way and helps avoid confusion. The primary exception to this you will see is in PR, where it is traditional not to use serial commas because it is often the style of journalists.

Commas are always important, considering the vast difference in meanings of the following two statements.

- Let's eat, Grandma
- Let's eat Grandma.

Correct punctuation could save Grandma's life.

If that does not convince you, maybe this will:



Contractions

Do not use contractions in Ansible documents.

Em dashes

When possible, use em-dashes with no space on either side. When full em-dashes aren't available, use double-dashes with no spaces on either side–like this.

A pair of em dashes can be used in place of commas to enhance readability. Note, however, that dashes are always more emphatic than commas.

A pair of em dashes can replace a pair of parentheses. Dashes are considered less formal than parentheses; they are also more intrusive. If you want to draw attention to the parenthetical content, use dashes. If you want to include the parenthetical content more subtly, use parentheses.

Note: When dashes are used in place of parentheses, surrounding punctuation should be omitted. Compare the following examples.

```
Upon discovering the errors (all 124 of them), the publisher immediately recalled the␣
↪books.

Upon discovering the errors-all 124 of them-the publisher immediately recalled the books.
```

When used in place of parentheses at the end of a sentence, only a single dash is used.

```
After three weeks on set, the cast was fed up with his direction (or, rather, lack of␣
↪direction).

After three weeks on set, the cast was fed up with his direction-or, rather, lack of␣
↪direction.
```

Exclamation points (!)

Do not use them at the end of sentences. An exclamation point can be used when referring to a command, such as the bang (!) command.

Gender References

Do not use gender-specific pronouns in documentation. It is far less awkward to read a sentence that uses "they" and "their" rather than "he/she" and "his/hers."

It is fine to use "you" when giving instructions and "the user," "new users," and so on. in more general explanations.

Never use "one" in place of "you" when writing technical documentation. Using "one" is far too formal.

Never use "we" when writing. "We" aren't doing anything on the user side. Ansible's products are doing the work as requested by the user.

Hyphen

The hyphen's primary function is the formation of certain compound terms. Do not use a hyphen unless it serves a purpose. If a compound adjective cannot be misread or, as with many psychological terms, its meaning is established, a hyphen is not necessary.

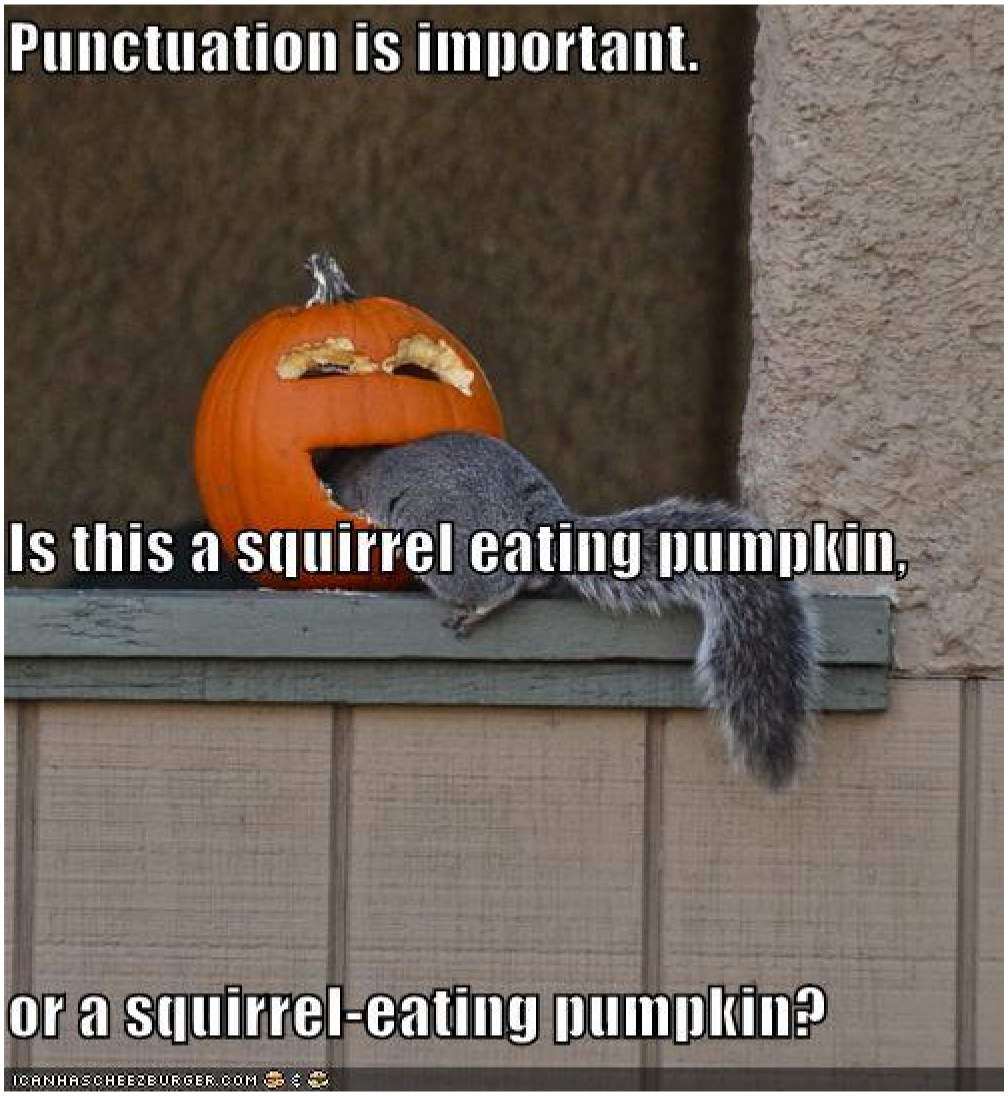
Use hyphens to avoid ambiguity or confusion:

```
a little-used car
a little used-car

cross complaint
cross-complaint

high-school girl
high schoolgirl
```

(continues on next page)

(continued from previous page)

```
fine-tooth comb (most people do not comb their teeth)

third-world war
third world war
```


Punctuation is important.
Is this a squirrel eating pumpkin,
or a squirrel-eating pumpkin?
ICANHASCHEEZBURGER.COM

In professionally printed material (particularly books, magazines, and newspapers), the hyphen is used to divide words between the end of one line and the beginning of the next. This allows for an evenly aligned right margin without highly variable (and distracting) word spacing.

Lists

Keep the structure of bulleted lists equivalent and consistent. If one bullet is a verb phrase, they should all be verb phrases. If one is a complete sentence, they should all be complete sentences, and so on.

Capitalize the first word of each bullet. Unless it is obvious that it is just a list of items, such as a list of items like: * computer * monitor * keyboard * mouse

When the bulleted list appears within the context of other copy, (unless it's a straight list like the previous example) add periods, even if the bullets are sentence fragments. Part of the reason behind this is that each bullet is said to complete the original sentence.

In some cases where the bullets are appearing independently, such as in a poster or a homepage promotion, they do not need periods.

When giving instructional steps, use numbered lists instead of bulleted lists.

Months and States

Abbreviate months and states according to AP. Months are only abbreviated if they are used in conjunction with a day. Example: "The President visited in January 1999." or "The President visited Jan. 12."

Months: Jan., Feb., March, April, May, June, July, Aug., Sept., Nov., Dec.

States: Ala., Ariz., Ark., Calif., Colo., Conn., Del., Fla., Ga., Ill., Ind., Kan., Ky., La., Md., Mass., Mich., Minn., Miss., Mo., Mont., Neb., Nev., NH, NJ, NM, NY, NC, ND, Okla., Ore., Pa., RI, SC, SD, Tenn., Vt., Va., Wash., W.Va., Wis., Wyo.

Numbers

Numbers between one and nine are written out. 10 and above are numerals. The exception to this is writing "4 million" or "4 GB." It's also acceptable to use numerals in tables and charts.

Phone Numbers

Phone number style: 1 (919) 555-0123 x002 and 1 888-GOTTEXT

Quotations (Using Quotation Marks and Writing Quotes)

> "Place the punctuation inside the quotes," the editor said.

Except in rare instances, use only "said" or "says" because anything else just gets in the way of the quote itself, and also tends to editorialize.

Place the name first right after the quote:
> "I like to write first-person because I like to become the character I'm writing," Wally Lamb said.

Not:
> "I like to write first-person because I like to become the character I'm writing," said Wally Lamb.

Semicolon

Use a semicolon to separate items in a series if the items contain commas:

- Everyday I have coffee, toast, and fruit for breakfast; a salad for lunch; and a peanut butter sandwich, cookies, ice cream, and chocolate cake for dinner.

Use a semicolon before a conjunctive adverb (however, therefore, otherwise, namely, for example, and so on): - I think; therefore, I am.

Spacing after sentences

Use only a single space after a sentence.

Time

- Time of day is written as "4 p.m."

Spelling - Word Usage - Common Words and Phrases to Use and Avoid

Acronyms

Always uppercase. An acronym is a word formed from the initial letters of a name, such as ROM for Read-only memory, SaaS for Software as a Service, or by combining initial letters or part of a series of words, such as LILO for LInux LOader.

Spell out the acronym before using it in alone text, such as "The Embedded DevKit (EDK)…"

Applications

When used as a proper name, use the capitalization of the product, such as GNUPro or Source-Navigator. When used as a command, use lowercase as appropriate, such as "To start GCC, type `gcc`."

Note: "vi" is always lowercase.

As

This is often used to mean "because", but has other connotations, for example, parallel or simultaneous actions. If you mean "because", say "because".

Asks for

Use “requests” instead.

Assure/Ensure/Insure

Assure implies a sort of mental comfort. As in “I assured my husband that I would eventually bring home beer.”

Ensure means “to make sure.”

Insure relates to monetary insurance.

Back up

This is a verb. You “back up” files; you do not “backup” files.

Backup

This is a noun. You create “backup” files; you do not create “back up” files.

Backward

Correct. Avoid using backwards unless you are stating that something has “backwards compatibility.”

Backwards compatibility

Correct as is.

By way of

Use “using” instead.

Can/May

Use “can” to describe actions or conditions that are possible. Use “may” only to describe situations where permission is being given. If either “can,” “could,” or “may” apply, use “can” because it’s less tentative.

CD or cd

When referring to a compact disk, use CD, such as “Insert the CD into the CD-ROM drive.” When referring to the change directory command, use cd.

CD-ROM

Correct. Do not use “cdrom,” “CD-Rom,” “CDROM,” “cd-rom” or any other variation. When referring to the drive, use CD-ROM drive, such as “Insert the CD into the CD-ROM drive.” The plural is “CD-ROMs.”

Command line

Correct. Do not use “command-line” or “commandline” as a noun. If used as an adjective, “command-line” is appropriate, for example “command-line arguments”.

Use “command line” to describes where to place options for a command, but not where to type the command. Use “shell prompt” instead to describe where to type commands. The line on the display screen where a command is expected. Generally, the command line is the line that contains the most recently displayed command prompt.

Daylight saving time (DST)

Correct. Do not use daylight savings time. Daylight Saving Time (DST) is often misspelled “Daylight Savings”, with an “s” at the end. Other common variations are “Summer Time”and “Daylight-Saving Time”. (https://www.timeanddate.com/time/dst/daylight-savings-time.html)

Download

Correct. Do not use “down load” or “down-load.”

e.g.

Spell it out: “For example.”

Failover

When used as a noun, a failover is a backup operation that automatically switches to a standby database, server or network if the primary system fails or is temporarily shut down for servicing. Failover is an important fault tolerance function of mission-critical systems that rely on constant accessibility. Failover automatically and transparently to the user redirects requests from the failed or down system to the backup system that mimics the operations of the primary system.

Fail over

When used as a verb, fail over is two words since there can be different tenses such as failed over.

Fewer

Fewer is used with plural nouns. Think things you could count. Time, money, distance, and weight are often listed as exceptions to the traditional "can you count it" rule, often thought of a singular amounts (the work will take less than 5 hours, for example).

File name

Correct. Do not use "filename."

File system

Correct. Do not use "filesystem." The system that an operating system or program uses to organize and keep track of files. For example, a hierarchical file system is one that uses directories to organize files into a tree structure. Although the operating system provides its own file management system, you can buy separate file management systems. These systems interact smoothly with the operating system but provide more features, such as improved backup procedures and stricter file protection.

For instance

For example," instead.

For further/additional/whatever information

Use "For more information"

For this reason

Use "therefore".

Forward

Correct. Avoid using "forwards."

Gigabyte (GB)

2 to the 30th power (1,073,741,824) bytes. One gigabyte is equal to 1,024 megabytes. Gigabyte is often abbreviated as G or GB.

Got

Avoid. Use “must” instead.

High-availability

Correct. Do not use “high availability.”

Highly available

Correct. Do not use highly-available.”

Hostname

Correct. Do not use host name.

i.e.

Spell it out: “That is.”

Installer

Avoid. Use “installation program” instead.

It’s and its

“It’s” is a contraction for “it is;” use “it is” instead of “it’s.” Use “its” as a possessive pronoun (for example, “the store is known for its low prices”).

Less

Less is used with singular nouns. For example “View less details” wouldn’t be correct but “View less detail” works. Use fewer when you have plural nouns (things you can count).

Linux

Correct. Do not use “LINUX” or “linux” unless referring to a command, such as “To start Linux, type linux.” Linux is a registered trademark of Linus Torvalds.

Login

A noun used to refer to the login prompt, such as “At the login prompt, enter your username.”

Log in

A verb used to refer to the act of logging in. Do not use “login,” “loggin,” “logon,” and other variants. For example, “When starting your computer, you are requested to log in. . . ”

Log on

To make a computer system or network recognize you so that you can begin a computer session. Most personal computers have no log-on procedure – you just turn the machine on and begin working. For larger systems and networks, however, you usually need to enter a username and password before the computer system will allow you to execute programs.

Lots of

Use “Several” or something equivalent instead.

Make sure

This means “be careful to remember, attend to, or find out something.” For example, “. . . make sure that the rhedk group is listed in the output.” Try to use verify or ensure instead.

Manual/man page

Correct. Two words. Do not use “manpage”

MB

(1) When spelled MB, short for megabyte (1,000,000 or 1,048,576 bytes, depending on the context).

(2) When spelled Mb, short for megabit.

MBps

Short for megabytes per second, a measure of data transfer speed. Mass storage devices are generally measured in MBps.

MySQL

Common open source database server and client package. Do not use “MYSQL” or “mySQL.”

Need to

Avoid. Use “must” instead.

Read-only

Correct. Use when referring to the access permissions of files or directories.

Real time/real-time

Depends. If used as a noun, it is the actual time during which something takes place. For example, “The computer may partly analyze the data in real time (as it comes in) – R. H. March.” If used as an adjective, “real-time” is appropriate. For example, “XEmacs is a self-documenting, customizable, extensible, real-time display editor.”

Refer to

Use to indicate a reference (within a manual or website) or a cross-reference (to another manual or documentation source).

See

Don’t use. Use “Refer to” instead.

Since

This is often used to mean “because”, but “since” has connotations of time, so be careful. If you mean “because”, say “because”.

Tells

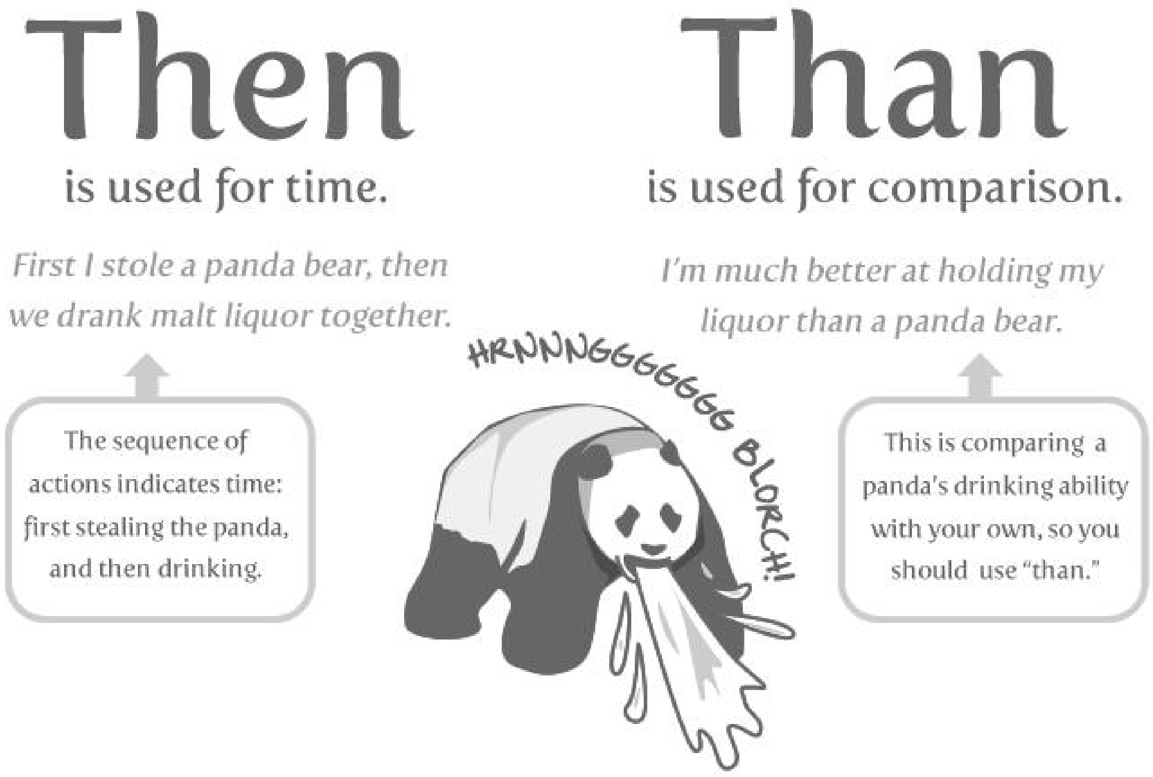
Use “Instructs” instead.

That/which

“That” introduces a restrictive clause-a clause that must be there for the sentence to make sense. A restrictive clause often defines the noun or phrase preceding it. “Which” introduces a non-restrictive, parenthetical clause-a clause that could be omitted without affecting the meaning of the sentence. For example: The car was travelling at a speed that would endanger lives. The car, which was traveling at a speed that would endanger lives, swerved onto the sidewalk. Use “who” or “whom,” rather than “that” or “which,” when referring to a person.

Then/than

"Then" refers to a time in the past or the next step in a sequence. "Than" is used for comparisons.


Then
is used for time.
First I stole a panda bear, then we drank malt liquor together.
The sequence of actions indicates time: first stealing the panda, and then drinking.
Than
is used for comparison.
I'm much better at holding my liquor than a panda bear.
This is comparing a panda's drinking ability with your own, so you should use "than."
HRNNNGGGGGGG BLORCH!

Third-party

Correct. Do not use "third party".

Troubleshoot

Correct. Do not use "trouble shoot" or "trouble-shoot." To isolate the source of a problem and fix it. In the case of computer systems, the term troubleshoot is usually used when the problem is suspected to be hardware -related. If the problem is known to be in software, the term debug is more commonly used.

UK

Correct as is, no periods.

UNIX®

Correct. Do not use “Unix” or “unix.” UNIX® is a registered trademark of The Open Group.

Unset

Don’t use. Use Clear.

US

Correct as is, no periods.

User

When referring to the reader, use “you” instead of “user.” For example, “The user must…” is incorrect. Use “You must…” instead. If referring to more than one user, calling the collection “users” is acceptable, such as “Other users may wish to access your database.”

Username

Correct. Do not use “user name.”

View

When using as a reference (“View the documentation available online.”), do not use View. Use “Refer to” instead.

Within

Don’t use to refer to a file that exists in a directory. Use “In”.

World Wide Web

Correct. Capitalize each word. Abbreviate as “WWW” or “Web.”

Webpage

Correct. Do not use “web page” or “Web page.”

Web server

Correct. Do not use “webserver”. For example, “The Apache HTTP Server is the default Web server…”

Website

Correct. Do not use “web site” or “Web site.” For example, “The Ansible website contains …”

Who/whom

Use the pronoun “who” as a subject. Use the pronoun “whom” as a direct object, an indirect object, or the object of a preposition. For example: Who owns this? To whom does this belong?

Will

Do not use future tense unless it is absolutely necessary. For instance, do not use the sentence, “The next section will describe the process in more detail.” Instead, use the sentence, “The next section describes the process in more detail.”

Wish

Use “need” instead of “desire” and “wish.” Use “want” when the reader’s actions are optional (that is, they may not “need” something but may still “want” something).

x86

Correct. Do not capitalize the “x.”

x86_64

Do not use. Do not use “Hammer”. Always use “AMD64 and Intel® EM64T” when referring to this architecture.

You

Correct. Do not use “I,” “he,” or “she.”

You may

Try to avoid using this. For example, “you may” can be eliminated from this sentence “You may double-click on the desktop…”

Writing documentation so search can find it

One of the keys to writing good documentation is to make it findable. Readers use a combination of internal site search and external search engines such as Google or duckduckgo.

To ensure Ansible documentation is findable, you should:

1. Use headings that clearly reflect what you are documenting.
2. Use numbered lists for procedures or high-level steps where possible.
3. Avoid linking to github blobs where possible.

Using clear headings in documentation

We all use simple English when we want to find something. For example, the title of this page could have been any one of the following:

- Search optimization
- Findable documentation
- Writing for findability

What we are really trying to describe is - how do I write documentation so search engines can find my content? That simple phrase is what drove the title of this section. When you are creating your headings for documentation, spend some time to think about what you would type in a search box to find it, or more importantly, how someone less familiar with Ansible would try to find that information. Your heading should be the answer to that question.

One word of caution - you do want to limit the size of your headings. A full heading such as *How do I write documentation so search engines can find my content?* is too long. Search engines would truncate anything over 50 - 60 characters. Long headings would also wrap on smaller devices such as a smart phone.

Using numbered lists for *zero position* snippets

Google can optimize the search results by adding a feature snippet at the top of the search results. This snippet provides a small window into the documentation on that first search result that adds more detail than the rest of the search results, and can occasionally answer the reader's questions right there, or at least verify that the linked page is what the reader is looking for.

Google returns the feature snippet in the form of numbered steps. Where possible, you should add a numbered list near the top of your documentation page, where appropriate. The steps can be the exact procedure a reader would follow, or could be a high level introduction to the documentation topic, such as the numbered list at the top of this page.

Problems with github blobs on search results

Search engines do not typically return github blobs in search results, at least not in higher ranked positions. While it is possible and sometimes necessary to link to github blobs from documentation, the better approach would be to copy that information into an .rst page in Ansible documentation.

Other search hints

While it may not be possible to adapt your documentation to all search optimizations, keep the following in mind as you write your documentation:

- **Search engines don't parse beyond the `#` in an html page.** So for example, all the subheadings on this page are appended to the main page URL. As such, when I search for 'Using number lists for zero position snippets', the search result would be a link to the top of this page, not a link directly to the subheading I searched for. Using *local TOCs* helps alleviate this problem as the reader can scan for the header at top of the page and click to the section they are looking for. For critical documentation, consider creating a new page that can be a direct search result page.
- **Make your first few sentences clearly describe your page topic.** Search engines return not just the URL, but a short description of the information at the URL. For Ansible documentation, we do not have description metadata embedded on each page. Instead, the search engines return the first couple of sentences (140 characters) on the page. That makes your first sentence or two very important to the reader who is searching for something in Ansible.

Resources

- Follow the style of the *Ansible Documentation*
- Ask for advice on the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat)
- Review these online style guides:
  - AP Stylebook
  - Chicago Manual of Style
  - Strunk and White's Elements of Style
  - Google developer documentation style guide

See also:

Contributing to the Ansible Documentation
How to contribute to the Ansible documentation

Testing the documentation locally
How to build the Ansible documentation

irc.libera.chat
#ansible-docs IRC chat channel

1.17 Developer Guide

Note: Making Open Source More Inclusive

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. We ask that you open an issue or pull request if you come upon a term that we have missed. For more details, see our CTO Chris Wright's message.

Welcome to the Ansible Developer Guide!

Who should use this guide?

If you want to extend Ansible by using a custom module or plugin locally, creating a module or plugin, adding functionality to an existing module, or expanding test coverage, this guide is for you. We've included detailed information for developers on how to test and document modules, as well as the prerequisites for getting your module or plugin accepted into the main Ansible repository.

Find the task that best describes what you want to do:

- I'm looking for a way to address a use case:
  - I want to *add a custom plugin or module locally*.
  - I want to figure out if *developing a module is the right approach* for my use case.
  - I want to *develop a collection*.
  - I want to *contribute to an Ansible-maintained collection*.
  - I want to *contribute to a community-maintained collection*.
  - I want to *migrate a role to a collection*.
- I've read the info above, and I'm sure I want to develop a module:
  - What do I need to know before I start coding?
  - I want to *set up my Python development environment*.
  - I want to *get started writing a module*.
  - **I want to write a specific kind of module:**
    - a *network module*
    - a *Windows module*.
    - an Amazon module.
    - an oVirt/RHV module.
    - a VMware module.
  - I want to *write a series of related modules* that integrate Ansible with a new product (for example, a database, cloud provider, network platform, and so on).
- I want to refine my code:
  - I want to *debug my module code*.
  - I want to *add tests*.
  - I want to *document my module*.
  - I want to *document my set of modules for a network platform*.
  - I want to follow *conventions and tips for clean, usable module code*.
  - I want to *make sure my code runs on Python 2 and Python 3*.
- I want to work on other development projects:
  - I want to *write a plugin*.
  - I want to *connect Ansible to a new source of inventory*.
  - I want to *deprecate an outdated module*.
- I want to contribute back to the Ansible project:
  - I want to *understand how to contribute to Ansible*.

  - I want to *contribute my module or plugin*.
  - I want to *understand the license agreement* for contributions to Ansible.

If you prefer to read the entire guide, here's a list of the pages in order.

1.17.1 Adding modules and plugins locally

You can extend Ansible by adding custom modules or plugins. You can create them from scratch or copy existing ones for local use. You can store a local module or plugin on your Ansible control node and share it with your team or organization. You can also share plugins and modules by including them in a collection, then publishing the collection on Ansible Galaxy.

If you are using a local module or plugin but Ansible cannot find it, this page is all you need.

If you want to create a plugin or a module, see *Developing plugins*, *Developing modules* and *Developing collections*.

Extending Ansible with local modules and plugins offers shortcuts such as:

- You can copy other people's modules and plugins.
- When writing a new module, you can choose any programming language you like.
- You do not have to clone any repositories.
- You do not have to open a pull request.
- You do not have to add tests (though we recommend that you do!).

Modules and plugins: what is the difference?

If you are looking to add functionality to Ansible, you might wonder whether you need a module or a plugin. Here is a quick overview to help you understand what you need:

- *Plugins* extend Ansible's core functionality. Most plugin types execute on the control node within the `/usr/bin/ansible` process. Plugins offer options and extensions for the core features of Ansible: transforming data, logging output, connecting to inventory, and more.
- Modules are a type of plugin that execute automation tasks on a 'target' (usually a remote system). Modules work as standalone scripts that Ansible executes in their own process outside of the controller. Modules interface with Ansible mostly via JSON, accepting arguments and returning information by printing a JSON string to stdout before exiting. Unlike the other plugins (which must be written in Python), modules can be written in any language; although Ansible provides modules in Python and Powershell only.

Adding modules and plugins in collections

You can add modules and plugins by *creating a collection*. With a collection, you can use custom modules and plugins in any playbook or role. You can share your collection easily at any time through Ansible Galaxy.

The rest of this page describes other methods of using local, standalone modules or plugins.

Adding a module or plugin outside of a collection

You can configure Ansible to load standalone local modules or plugins in specific locations and make them available to all playbooks and roles (using configured paths). Alternatively, you can make a non-collection local module or plugin available only to certain playbooks or roles (via adjacent paths).

Adding standalone local modules for all playbooks and roles

To load standalone local modules automatically and make them available to all playbooks and roles, use the *DEFAULT_MODULE_PATH* configuration setting or the `ANSIBLE_LIBRARY` environment variable. The configuration setting and environment variable take a colon-separated list, similar to `$PATH`. You have two options:

- Add your standalone local module to one of the default configured locations. See the *DEFAULT_MODULE_PATH* configuration setting for details. Default locations may change without notice.
- **Add the location of your standalone local module to an environment variable or configuration:**
    - the `ANSIBLE_LIBRARY` environment variable
    - the *DEFAULT_MODULE_PATH* configuration setting

To view your current configuration settings for modules:

```
ansible-config dump |grep DEFAULT_MODULE_PATH
```

After you save your module file in one of these locations, Ansible loads it and you can use it in any local task, playbook, or role.

To confirm that `my_local_module` is available:

- type `ansible localhost -m my_local_module` to see the output for that module, or
- type `ansible-doc -t module my_local_module` to see the documentation for that module

Note: This applies to all plugin types but requires specific configuration and/or adjacent directories for each plugin type, see below.

Note: The `ansible-doc` command can parse module documentation from modules written in Python or an adjacent YAML file. If you have a module written in a programming language other than Python, you should write the documentation in a Python or YAML file adjacent to the module file. *Adjacent YAML documentation files*

Adding standalone local modules for selected playbooks or a single role

Ansible automatically loads all executable files from certain directories adjacent to your playbook or role as modules. Standalone modules in these locations are available only to the specific playbook, playbooks, or role in the parent directory.

- To use a standalone module only in a selected playbook or playbooks, store the module in a subdirectory called `library` in the directory that contains the playbook or playbooks.
- To use a standalone module only in a single role, store the module in a subdirectory called `library` within that role.

Note: This applies to all plugin types but requires specific configuration and/or adjacent directories for each plugin type, see below.

Warning: Roles contained in collections cannot contain any modules or other plugins. All plugins in a collection must live in the collection `plugins` directory tree. All plugins in that tree are accessible to all roles in the collection. If you are developing new modules, we recommend distributing them in *collections*, not in roles.

Adding a non-module plugin locally outside of a collection

You can configure Ansible to load standalone local plugins in a specified location or locations and make them available to all playbooks and roles. Alternatively, you can make a standalone local plugin available only to specific playbooks or roles.

Note: Although modules are plugins, the naming patterns for directory names and environment variables that apply to other plugin types do not apply to modules. See *Adding a module or plugin outside of a collection*.

Adding local non-module plugins for all playbooks and roles

To load standalone local plugins automatically and make them available to all playbooks and roles, use the configuration setting or environment variable for the type of plugin you are adding. These configuration settings and environment variables take a colon-separated list, similar to `$PATH`. You have two options:

- Add your local plugin to one of the default configured locations. See *configuration settings* for details on the correct configuration setting for the plugin type. Default locations may change without notice.
- **Add the location of your local plugin to an environment variable or configuration:**
  - the relevant `ANSIBLE_plugin_type_PLUGINS` environment variable - for example, `$ANSIBLE_INVENTORY_PLUGINS` or `$ANSIBLE_VARS_PLUGINS`
  - the relevant `plugin_type_PATH` configuration setting, most of which begin with `DEFAULT_` - for example, `DEFAULT_CALLBACK_PLUGIN_PATH` or `DEFAULT_FILTER_PLUGIN_PATH` or `BECOME_PLUGIN_PATH`

To view your current configuration settings for non-module plugins:

```
ansible-config dump |grep plugin_type_PATH
```

After your plugin file is added to one of these locations, Ansible loads it and you can use it in any local module, task, playbook, or role. For more information on environment variables and configuration settings, see *Ansible Configuration Settings*.

To confirm that `plugins/plugin_type/my_local_plugin` is available:

- type `ansible-doc -t <plugin_type> my_local_lookup_plugin` to see the documentation for that plugin - for example, `ansible-doc -t lookup my_local_lookup_plugin`

The `ansible-doc` command works for most plugin types, but not for action, filter, or test plugins. See *ansible-doc* for more details.

Adding standalone local plugins for selected playbooks or a single role

Ansible automatically loads all plugins from certain directories adjacent to your playbook or role, loading each type of plugin separately from a directory named for the type of plugin. Standalone plugins in these locations are available only to the specific playbook, playbooks, or role in the parent directory.

- To use a standalone plugin only in a selected playbook or playbooks, store the plugin in a subdirectory for the correct `plugin_type` (for example, `callback_plugins` or `inventory_plugins`) in the directory that contains the playbooks. These directories must use the `_plugins` suffix. For a full list of plugin types, see *Working with plugins*.
- To use a standalone plugin only in a single role, store the plugin in a subdirectory for the correct `plugin_type` (for example, `cache_plugins` or `strategy_plugins`) within that role. When shipped as part of a role, the plugin is available as soon as the role is executed. These directories must use the `_plugins` suffix. For a full list of plugin types, see *Working with plugins*.

Warning: Roles contained in collections cannot contain any plugins. All plugins in a collection must live in the collection `plugins` directory tree. All plugins in that tree are accessible to all roles in the collection. If you are developing new plugins, we recommend distributing them in *collections*, not in roles.

Using `ansible.legacy` to access custom versions of an `ansible.builtin` module

If you need to override one of the `ansible.builtin` modules and are using FQCN, you need to use `ansible.legacy` as part of the fully-qualified collection name (FQCN). For example, if you had your own `copy` module, you would access it as `ansible.legacy.copy`. See *Using ansible.legacy to access local custom modules from collections-based roles* for details on how to use custom modules with roles within a collection.

1.17.2 Should you develop a module?

Developing Ansible modules is easy, but often it is not necessary. Before you start writing a new module, ask:

1. Does a similar module already exist?

An existing module may cover the functionality you want. Ansible collections include thousands of modules. Search our list of included collections or Ansible Galaxy to see if an existing module does what you need.

2. Should you use or develop an action plugin instead of a module?

An action plugin may be the best way to get the functionality you want. Action plugins run on the control node instead of on the managed node, and their functionality is available to all modules. For more information about developing plugins, read the *developing plugins page*.

3. Should you use a role instead of a module?

A combination of existing modules may cover the functionality you want. You can write a role for this type of use case. Check out the *roles documentation*.

4. Should you create a collection instead of a single module?

The functionality you want may be too large for a single module. If you want to connect Ansible to a new cloud provider, database, or network platform, you may need to *develop a new collection*.

- Each module should have a concise and well defined functionality. Basically, follow the UNIX philosophy of doing one thing well.
- A module should not require that a user know all the underlying options of an API/tool to be used. For instance, if the legal values for a required module parameter cannot be documented, that's a sign that the module would be rejected.
- Modules should typically encompass much of the logic for interacting with a resource. A lightweight wrapper around an API that does not contain much logic would likely cause users to offload too much logic into a playbook, and for this reason the module would be rejected. Instead try creating multiple modules for interacting with smaller individual pieces of the API.

If your use case isn't covered by an existing module, an action plugin, or a role, and you don't need to create multiple modules, then you're ready to start developing a new module. Choose from the topics below for next steps:

- I want to *get started on a new module*.
- I want to review *tips and conventions for developing good modules*.
- I want to *write a Windows module*.
- I want *an overview of Ansible's architecture*.
- I want to *document my module*.
- I want to *contribute my module to an existing Ansible collection*.
- I want to *add unit and integration tests to my module*.
- I want to *add Python 3 support to my module*.
- I want to *write multiple modules*.

See also:

Collection Index
Browse existing collections, modules, and plugins

Mailing List
Development mailing list

Real-time chat
How to join Ansible chat channels

1.17.3 Developing modules

A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely. Modules interact with your local machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance. Each module can be used by the Ansible API, or by the **ansible** or **ansible-playbook** programs. A module provides a defined interface, accepts arguments, and returns information to Ansible by printing a JSON string to stdout before exiting.

If you need functionality that is not available in any of the thousands of Ansible modules found in collections, you can easily write your own custom module. When you write a module for local use, you can choose any programming language and follow your own rules. Use this topic to learn how to create an Ansible module in Python. After you create a module, you must add it locally to the appropriate directory so that Ansible can find and execute it. For details about adding a module locally, see *Adding modules and plugins locally*.

If you are developing a module in a *collection*, see those documents instead.

Preparing an environment for developing Ansible modules

You just need `ansible-core` installed to test the module. Modules can be written in any language, but most of the following guide is assuming you are using Python. Modules for inclusion in Ansible itself must be Python or Powershell.

One advantage of using Python or Powershell for your custom modules is being able to use the `module_utils` common code that does a lot of the heavy lifting for argument processing, logging and response writing, among other things.

Creating a module

It is highly recommended that you use a `venv` or `virtualenv` for Python development.

To create a module:

1. Create a `library` directory in your workspace, your test play should live in the same directory.
2. Create your new module file: `$ touch library/my_test.py`. Or just open/create it with your editor of choice.
3. Paste the content below into your new module file. It includes the *required Ansible format and documentation*, a simple *argument spec for declaring the module options*, and some example code.
4. Modify and extend the code to do what you want your new module to do. See the *programming tips* and *Python 3 compatibility* pages for pointers on writing clean and concise module code.

```
#!/usr/bin/python

# Copyright: (c) 2018, Terry Jones <terry.jones@example.org>
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.
↪txt)
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

```

(continues on next page)

(continued from previous page)

```
DOCUMENTATION = r'''
---
module: my_test

short_description: This is my test module

# If this is part of a collection, you need to use semantic versioning,
# i.e. the version is of the form "2.5.0" and not "2.4".
version_added: "1.0.0"

description: This is my longer description explaining my test module.

options:
    name:
        description: This is the message to send to the test module.
        required: true
        type: str
    new:
        description:
            - Control to demo if the result of this module is changed or not.
            - Parameter description can be a list as well.
        required: false
        type: bool
# Specify this value according to your collection
# in format of namespace.collection.doc_fragment_name
extends_documentation_fragment:
    - my_namespace.my_collection.my_doc_fragment_name

author:
    - Your Name (@yourGitHubHandle)
'''

EXAMPLES = r'''
# Pass in a message
- name: Test with a message
  my_namespace.my_collection.my_test:
    name: hello world

# pass in a message and have changed true
- name: Test with a message and changed output
  my_namespace.my_collection.my_test:
    name: hello world
    new: true

# fail the module
- name: Test failure of the module
  my_namespace.my_collection.my_test:
    name: fail me
'''

RETURN = r'''
# These are examples of possible return values, and in general should use other names␣
```

(continues on next page)

(continued from previous page)

```
↪for return values.
original_message:
    description: The original name param that was passed in.
    type: str
    returned: always
    sample: 'hello world'
message:
    description: The output message that the test module generates.
    type: str
    returned: always
    sample: 'goodbye'
'''

from ansible.module_utils.basic import AnsibleModule


def run_module():
    # define available arguments/parameters a user can pass to the module
    module_args = dict(
        name=dict(type='str', required=True),
        new=dict(type='bool', required=False, default=False)
    )

    # seed the result dict in the object
    # we primarily care about changed and state
    # changed is if this module effectively modified the target
    # state will include any data that you want your module to pass back
    # for consumption, for example, in a subsequent task
    result = dict(
        changed=False,
        original_message='',
        message=''
    )

    # the AnsibleModule object will be our abstraction working with Ansible
    # this includes instantiation, a couple of common attr would be the
    # args/params passed to the execution, as well as if the module
    # supports check mode
    module = AnsibleModule(
        argument_spec=module_args,
        supports_check_mode=True
    )

    # if the user is working with this module in only check mode we do not
    # want to make any changes to the environment, just return the current
    # state with no modifications
    if module.check_mode:
        module.exit_json(**result)

    # manipulate or modify the state as needed (this is going to be the
    # part where your module will do what it needs to do)
    result['original_message'] = module.params['name']
```

(continues on next page)

(continued from previous page)

```
    result['message'] = 'goodbye'

    # use whatever logic you need to determine whether or not this module
    # made any modifications to your target
    if module.params['new']:
        result['changed'] = True

    # during the execution of the module, if there is an exception or a
    # conditional state that effectively causes a failure, run
    # AnsibleModule.fail_json() to pass in the message and the result
    if module.params['name'] == 'fail me':
        module.fail_json(msg='You requested this to fail', **result)

    # in the event of a successful module execution, you will want to
    # simple AnsibleModule.exit_json(), passing the key/value results
    module.exit_json(**result)


def main():
    run_module()


if __name__ == '__main__':
    main()
```

Creating an info or a facts module

Ansible gathers information about the target machines using facts modules, and gathers information on other objects or files using info modules. If you find yourself trying to add `state: info` or `state: list` to an existing module, that is often a sign that a new dedicated `_facts` or `_info` module is needed.

In Ansible 2.8 and onwards, we have two type of information modules, they are `*_info` and `*_facts`.

If a module is named `<something>_facts`, it should be because its main purpose is returning `ansible_facts`. Do not name modules that do not do this with `_facts`. Only use `ansible_facts` for information that is specific to the host machine, for example network interfaces and their configuration, which operating system and which programs are installed.

Modules that query/return general information (and not `ansible_facts`) should be named `_info`. General information is non-host specific information, for example information on online/cloud services (you can access different accounts for the same online service from the same host), or information on VMs and containers accessible from the machine, or information on individual files or programs.

Info and facts modules, are just like any other Ansible Module, with a few minor requirements:

1. They MUST be named `<something>_info` or `<something>_facts`, where <something> is singular.
2. Info `*_info` modules MUST return in the form of the *result dictionary* so other modules can access them.
3. Fact `*_facts` modules MUST return in the `ansible_facts` field of the *result dictionary* so other modules can access them.
4. They MUST support *check_mode*.
5. They MUST NOT make any changes to the system.

6. They MUST document the *return fields* and *examples*.

The rest is just like creating a normal module.

Verifying your module code

After you modify the sample code above to do what you want, you can try out your module. Our *debugging tips* will help if you run into bugs as you verify your module code.

Verifying your module code locally

The simplest way is to use `ansible` adhoc command:

```
ANSIBLE_LIBRARY=./library ansible -m my_test -a 'name=hello new=true' remotehost
```

If your module does not need to target a remote host, you can quickly and easily exercise your code locally like this:

```
ANSIBLE_LIBRARY=./library ansible -m my_test -a 'name=hello new=true' localhost
```

- If for any reason (pdb, using print(), faster iteration, etc) you want to avoid going through Ansible, another way is to create an arguments file, a basic JSON config file that passes parameters to your module so that you can run it. Name the arguments file `/tmp/args.json` and add the following content:

```
{
    "ANSIBLE_MODULE_ARGS": {
        "name": "hello",
        "new": true
    }
}
```

- Then the module can be tested locally and directly. This skips the packing steps and uses module_utils files directly:

```
``$ python library/my_test.py /tmp/args.json``
```

It should return output like this:

```
{"changed": true, "state": {"original_message": "hello", "new_message": "goodbye"},
↪"invocation": {"module_args": {"name": "hello", "new": true}}}
```

Verifying your module code in a playbook

You can easily run a full test by including it in a playbook, as long as the `library` directory is in the same directory as the play:

- Create a playbook in any directory: `$ touch testmod.yml`
- Add the following to the new playbook file:

```
- name: test my new module
  hosts: localhost
  tasks:
  - name: run the new module
```

(continues on next page)

(continued from previous page)

```
    my_test:
      name: 'hello'
      new: true
    register: testout
  - name: dump test output
    debug:
      msg: '{{ testout }}'
```

- Run the playbook and analyze the output: `$ ansible-playbook ./testmod.yml`

Testing your newly-created module

The following two examples will get you started with testing your module code. Please review our *testing* section for more detailed information, including instructions for testing module documentation, adding integration tests, and more.

Note: If contributing to Ansible, every new module and plugin should have integration tests, even if the tests cannot be run on Ansible CI infrastructure. In this case, the tests should be marked with the `unsupported` alias in aliases file.

Performing sanity tests

You can run through Ansible's sanity checks in a container:

`$ ansible-test sanity -v --docker --python 3.10 MODULE_NAME`

Note: Note that this example requires Docker to be installed and running. If you'd rather not use a container for this, you can choose to use `--venv` instead of `--docker`.

Contributing back to Ansible

If you would like to contribute to `ansible-core` by adding a new feature or fixing a bug, create a fork of the ansible/ansible repository and develop against a new feature branch using the `devel` branch as a starting point. When you have a good working code change, you can submit a pull request to the Ansible repository by selecting your feature branch as a source and the Ansible devel branch as a target.

If you want to contribute a module to an *Ansible collection*, review our *submission checklist*, *programming tips*, and *strategy for maintaining Python 2 and Python 3 compatibility*, as well as information about *testing* before you open a pull request.

The *Community Guide* covers how to open a pull request and what happens next.

Communication and development support

Join the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat) for discussions surrounding Ansible development.

For questions and discussions pertaining to using the Ansible product, join the `#ansible` channel.

To find other topic-specific chat channels, look at *Community Guide, Communicating*.

Credit

Thank you to Thomas Stringer (@trstringer) for contributing source material for this topic.

1.17.4 Contributing your module to an existing Ansible collection

If you want to contribute a module to an existing collection, you must meet the community's objective and subjective requirements. Please read the details below, and also review our *tips for module development*.

Modules accepted into certain collections are included in every Ansible release on PyPI. However, contributing to one of these collections is not the only way to distribute a module - you can *create your own collection*, embed modules in roles on Galaxy or simply share copies of your module code for *local use*.

Contributing modules: objective requirements

To contribute a module to most Ansible collections, you must:

- write your module in either Python or Powershell for Windows
- use the `AnsibleModule` common code
- support Python 2.6 and Python 3.5 - if your module cannot support Python 2.6, explain the required minimum Python version and rationale in the requirements section in `DOCUMENTATION`
- use proper *Python 3 syntax*
- follow PEP 8 Python style conventions - see testing_pep8 for more information
- license your module under the GPL license (GPLv3 or later)
- understand the *license agreement*, which applies to all contributions
- conform to Ansible's *formatting and documentation* standards
- include comprehensive *tests* for your module
- minimize module dependencies
- support *check_mode* if possible
- ensure your code is readable
- if a module is named `<something>_facts`, it should be because its main purpose is returning `ansible_facts`. Do not name modules that do not do this with `_facts`. Only use `ansible_facts` for information that is specific to the host machine, for example network interfaces and their configuration, which operating system and which programs are installed.
- Modules that query/return general information (and not `ansible_facts`) should be named `_info`. General information is non-host specific information, for example information on online/cloud services (you can access different accounts for the same online service from the same host), or information on VMs and containers accessible from the machine.

Additional requirements may apply for certain collections. Review the individual collection repositories for more information.

Please make sure your module meets these requirements before you submit your PR/proposal. If you have questions, reach out by using the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat) or the Ansible development mailing list.

Contributing to Ansible: subjective requirements

If your module meets these objective requirements, collection maintainers will review your code to see if they think it's clear, concise, secure, and maintainable. They will consider whether your module provides a good user experience, helpful error messages, reasonable defaults, and more. This process is subjective, with no exact standards for acceptance. For the best chance of getting your module accepted, follow our *tips for module development*.

Other checklists

- *Tips for module development*.
- *Windows development checklist*.

1.17.5 Conventions, tips, and pitfalls

Topics

As you design and develop modules, follow these basic conventions and tips for clean, usable code:

Scoping your module(s)

Especially if you want to contribute your module(s) to an existing Ansible Collection, make sure each module includes enough logic and functionality, but not too much. If these guidelines seem confusing, consider *whether you really need to write a module* at all.

- Each module should have a concise and well-defined functionality. Basically, follow the UNIX philosophy of doing one thing well.
- Do not add `get`, `list` or `info` state options to an existing module - create a new `_info` or `_facts` module.
- Modules should not require that a user know all the underlying options of an API/tool to be used. For instance, if the legal values for a required module option cannot be documented, the module does not belong in Ansible Core.
- Modules should encompass much of the logic for interacting with a resource. A lightweight wrapper around a complex API forces users to offload too much logic into their playbooks. If you want to connect Ansible to a complex API, *create multiple modules* that interact with smaller individual pieces of the API.
- Avoid creating a module that does the work of other modules; this leads to code duplication and divergence, and makes things less uniform, unpredictable and harder to maintain. Modules should be the building blocks. If you are asking 'how can I have a module execute other modules' … you want to write a role.

Designing module interfaces

- If your module is addressing an object, the option for that object should be called `name` whenever possible, or accept `name` as an alias.
- Modules accepting boolean status should accept `yes`, `no`, `true`, `false`, or anything else a user may likely throw at them. The AnsibleModule common code supports this with `type='bool'`.
- Avoid `action/command`, they are imperative and not declarative, there are other ways to express the same thing.

General guidelines & tips

- Each module should be self-contained in one file, so it can be auto-transferred by `ansible-core`.
- Module name MUST use underscores instead of hyphens or spaces as a word separator. Using hyphens and spaces will prevent `ansible-core` from importing your module.
- Always use the `hacking/test-module.py` script when developing modules - it will warn you about common pitfalls.
- If you have a local module that returns information specific to your installations, a good name for this module is `site_info`.
- Eliminate or minimize dependencies. If your module has dependencies, document them at the top of the module file and raise JSON error messages when dependency import fails.
- Don't write to files directly; use a temporary file and then use the `atomic_move` function from `ansible.module_utils.basic` to move the updated temporary file into place. This prevents data corruption and ensures that the correct context for the file is kept.
- Avoid creating caches. Ansible is designed without a central server or authority, so you cannot guarantee it will not run with different permissions, options or locations. If you need a central authority, have it on top of Ansible (for example, using bastion/cm/ci server, AWX, or the Red Hat Ansible Automation Platform); do not try to build it into modules.
- If you package your module(s) in an RPM, install the modules on the control machine in `/usr/share/ansible`. Packaging modules in RPMs is optional.

Functions and Methods

- Each function should be concise and should describe a meaningful amount of work.
- "Don't repeat yourself" is generally a good philosophy.
- Function names should use underscores: `my_function_name`.
- The name of each function should describe what the function does.
- Each function should have a docstring.
- If your code is too nested, that's usually a sign the loop body could benefit from being a function. Parts of our existing code are not the best examples of this at times.

Python tips

- Include a `main` function that wraps the normal execution.
- Call your `main` function from a conditional so you can import it into unit tests - for example:

```
if __name__ == '__main__':
    main()
```

Importing and using shared code

- Use shared code whenever possible - don't reinvent the wheel. Ansible offers the `AnsibleModule` common Python code, plus *utilities* for many common use cases and patterns. You can also create documentation fragments for docs that apply to multiple modules.
- Import `ansible.module_utils` code in the same place as you import other libraries.
- Do NOT use wildcards (*) for importing other python modules; instead, list the function(s) you are importing (for example, `from some.other_python_module.basic import otherFunction`).
- Import custom packages in `try`/`except`, capture any import errors, and handle them with `fail_json()` in `main()`. For example:

```
import traceback

from ansible.module_utils.basic import missing_required_lib

LIB_IMP_ERR = None
try:
    import foo
    HAS_LIB = True
except:
    HAS_LIB = False
    LIB_IMP_ERR = traceback.format_exc()
```

Then in `main()`, just after the argspec, do

```
if not HAS_LIB:
    module.fail_json(msg=missing_required_lib("foo"),
                     exception=LIB_IMP_ERR)
```

And document the dependency in the `requirements` section of your module's *DOCUMENTATION block*.

Handling module failures

When your module fails, help users understand what went wrong. If you are using the `AnsibleModule` common Python code, the `failed` element will be included for you automatically when you call `fail_json`. For polite module failure behavior:

- Include a key of `failed` along with a string explanation in `msg`. If you don't do this, Ansible will use standard return codes: 0=success and non-zero=failure.
- Don't raise a traceback (stacktrace). Ansible can deal with stacktraces and automatically converts anything unparsable into a failed result, but raising a stacktrace on module failure is not user-friendly.
- Do not use `sys.exit()`. Use `fail_json()` from the module object.

Handling exceptions (bugs) gracefully

- Validate upfront–fail fast and return useful and clear error messages.
- Use defensive programming–use a simple design for your module, handle errors gracefully, and avoid direct stacktraces.
- Fail predictably–if we must fail, do it in a way that is the most expected. Either mimic the underlying tool or the general way the system works.
- Give out a useful message on what you were doing and add exception messages to that.
- Avoid catchall exceptions, they are not very useful unless the underlying API gives very good error messages pertaining the attempted action.

Creating correct and informative module output

Modules must output valid JSON only. Follow these guidelines for creating correct, useful module output:

- Module return data must be encoded as strict UTF-8. Modules that cannot return UTF-8 encoded data should return the data encoded by something such as base64. Optionally modules can make the determination if they can encode as UTF-8 and utilize `errors='replace'` to replace non UTF-8 characters making the return values lossy.
- Make your top-level return type a hash (dictionary).
- Nest complex return values within the top-level hash.
- Incorporate any lists or simple scalar values within the top-level return hash.
- Do not send module output to standard error, because the system will merge standard out with standard error and prevent the JSON from parsing.
- Capture standard error and return it as a variable in the JSON on standard out. This is how the command module is implemented.
- Never do `print("some status message")` in a module, because it will not produce valid JSON output.
- Always return useful data, even when there is no change.
- Be consistent about returns (some modules are too random), unless it is detrimental to the state/action.
- Make returns reusable–most of the time you don't want to read it, but you do want to process it and re-purpose it.
- Return diff if in diff mode. This is not required for all modules, as it won't make sense for certain ones, but please include it when applicable.

- Enable your return values to be serialized as JSON with Python's standard JSON encoder and decoder library. Basic python types (strings, int, dicts, lists, and so on) are serializable.
- Do not return an object using exit_json(). Instead, convert the fields you need from the object into the fields of a dictionary and return the dictionary.
- Results from many hosts will be aggregated at once, so your module should return only relevant output. Returning the entire contents of a log file is generally bad form.

If a module returns stderr or otherwise fails to produce valid JSON, the actual output will still be shown in Ansible, but the command will not succeed.

Following Ansible conventions

Ansible conventions offer a predictable user interface across all modules, playbooks, and roles. To follow Ansible conventions in your module development:

- Use consistent names across modules (yes, we have many legacy deviations - don't make the problem worse!).
- Use consistent options (arguments) within your module(s).
- Do not use 'message' or 'syslog_facility' as an option name, because this is used internally by Ansible.
- Normalize options with other modules - if Ansible and the API your module connects to use different names for the same option, add aliases to your options so the user can choose which names to use in tasks and playbooks.
- Return facts from `*_facts` modules in the `ansible_facts` field of the *result dictionary* so other modules can access them.
- Implement `check_mode` in all `*_info` and `*_facts` modules. Playbooks which conditionalize based on fact information will only conditionalize correctly in `check_mode` if the facts are returned in `check_mode`. Usually you can add `supports_check_mode=True` when instantiating `AnsibleModule`.
- Use module-specific environment variables. For example, if you use the helpers in `module_utils.api` for basic authentication with `module_utils.urls.fetch_url()` and you fall back on environment variables for default values, use a module-specific environment variable like `API_<MODULENAME>_USERNAME` to avoid conflicts between modules.
- Keep module options simple and focused - if you're loading a lot of choices/states on an existing option, consider adding a new, simple option instead.
- Keep options small when possible. Passing a large data structure to an option might save us a few tasks, but it adds a complex requirement that we cannot easily validate before passing on to the module.
- If you want to pass complex data to an option, write an expert module that allows this, along with several smaller modules that provide a more 'atomic' operation against the underlying APIs and services. Complex operations require complex data. Let the user choose whether to reflect that complexity in tasks and plays or in vars files.
- Implement declarative operations (not CRUD) so the user can ignore existing state and focus on final state. For example, use `started/stopped`, `present/absent`.
- Strive for a consistent final state (aka idempotency). If running your module twice in a row against the same system would result in two different states, see if you can redesign or rewrite to achieve consistent final state. If you can't, document the behavior and the reasons for it.
- Provide consistent return values within the standard Ansible return structure, even if NA/None are used for keys normally returned under other options.

Module Security

- Avoid passing user input from the shell.
- Always check return codes.
- You must always use `module.run_command`, not `subprocess` or `Popen` or `os.system`.
- Avoid using the shell unless absolutely necessary.
- If you must use the shell, you must pass `use_unsafe_shell=True` to `module.run_command`.
- If any variables in your module can come from user input with `use_unsafe_shell=True`, you must wrap them with `pipes.quote(x)`.
- When fetching URLs, use `fetch_url` or `open_url` from `ansible.module_utils.urls`. Do not use `urllib2`, which does not natively verify TLS certificates and so is insecure for https.
- Sensitive values marked with `no_log=True` will automatically have that value stripped from module return values. If your module could return these sensitive values as part of a dictionary key name, you should call the `ansible.module_utils.basic.sanitize_keys()` function to strip the values from the keys. See the `uri` module for an example.

1.17.6 Ansible and Python 3

The `ansible-core` code runs Python 3 (for specific versions check *Control Node Requirements* Contributors to `ansible-core` and to Ansible Collections should be aware of the tips in this document so that they can write code that will run on the same versions of Python as the rest of Ansible.

We do have some considerations depending on the types of Ansible code:

1. controller-side code - code that runs on the machine where you invoke **`/usr/bin/ansible`**, only needs to support the controller's Python versions.
2. modules - the code which Ansible transmits to and invokes on the managed machine. Modules need to support the 'managed node' Python versions, with some exceptions.
3. shared `module_utils` code - the common code that is used by modules to perform tasks and sometimes used by controller-side code as well. Shared `module_utils` code needs to support the same range of Python as the modules.

However, the three types of code do not use the same string strategy. If you're developing a module or some `module_utils` code, be sure to read the section on string strategy carefully.

Minimum version of Python 3.x and Python 2.x

See *Control Node Requirements* and *Managed Node Requirements* for the specific versions supported.

Your custom modules can support any version of Python (or other languages) you want, but the above are the requirements for the code contributed to the Ansible project.

Developing Ansible code that supports Python 2 and Python 3

The best place to start learning about writing code that supports both Python 2 and Python 3 is Lennart Regebro's book: Porting to Python 3. The book describes several strategies for porting to Python 3. The one we're using is to support Python 2 and Python 3 from a single code base

Understanding strings in Python 2 and Python 3

Python 2 and Python 3 handle strings differently, so when you write code that supports Python 3 you must decide what string model to use. Strings can be an array of bytes (like in C) or they can be an array of text. Text is what we think of as letters, digits, numbers, other printable symbols, and a small number of unprintable "symbols" (control codes).

In Python 2, the two types for these (`str` for bytes and `unicode` for text) are often used interchangeably. When dealing only with ASCII characters, the strings can be combined, compared, and converted from one type to another automatically. When non-ASCII characters are introduced, Python 2 starts throwing exceptions due to not knowing what encoding the non-ASCII characters should be in.

Python 3 changes this behavior by making the separation between bytes (`bytes`) and text (`str`) more strict. Python 3 will throw an exception when trying to combine and compare the two types. The programmer has to explicitly convert from one type to the other to mix values from each.

In Python 3 it's immediately apparent to the programmer when code is mixing the byte and text types inappropriately, whereas in Python 2, code that mixes those types may work until a user causes an exception by entering non-ASCII input. Python 3 forces programmers to proactively define a strategy for working with strings in their program so that they don't mix text and byte strings unintentionally.

Ansible uses different strategies for working with strings in controller-side code, in :ref: *modules <module_string_strategy>*, and in *module_utils* code.

Controller string strategy: the Unicode Sandwich

Until recently `ansible-core` supported Python 2.x and followed this strategy, known as the Unicode Sandwich (named after Python 2's `unicode` text type). For Unicode Sandwich we know that at the border of our code and the outside world (for example, file and network IO, environment variables, and some library calls) we are going to receive bytes. We need to transform these bytes into text and use that throughout the internal portions of our code. When we have to send those strings back out to the outside world we first convert the text back into bytes. To visualize this, imagine a 'sandwich' consisting of a top and bottom layer of bytes, a layer of conversion between, and all text type in the center.

For compatibility reasons you will see a bunch of custom functions we developed (`to_text`/`to_bytes`/`to_native`) and while Python 2 is not a concern anymore we will continue to use them as they apply for other cases that make dealing with unicode problematic.

While we will not be using it most of it anymore, the documentation below is still useful for those developing modules that still need to support both Python 2 and 3 simultaneously.

Unicode Sandwich common borders: places to convert bytes to text in controller code

This is a partial list of places where we have to convert to and from bytes when using the Unicode Sandwich string strategy. It's not exhaustive but it gives you an idea of where to watch for problems.

Reading and writing to files

In Python 2, reading from files yields bytes. In Python 3, it can yield text. To make code that's portable to both we don't make use of Python 3's ability to yield text but instead do the conversion explicitly ourselves. For example:

```python
from ansible.module_utils.common.text.converters import to_text

with open('filename-with-utf8-data.txt', 'rb') as my_file:
    b_data = my_file.read()
    try:
        data = to_text(b_data, errors='surrogate_or_strict')
    except UnicodeError:
        # Handle the exception gracefully -- usually by displaying a good
        # user-centric error message that can be traced back to this piece
        # of code.
        pass
```

Note: Much of Ansible assumes that all encoded text is UTF-8. At some point, if there is demand for other encodings we may change that, but for now it is safe to assume that bytes are UTF-8.

Writing to files is the opposite process:

```python
from ansible.module_utils.common.text.converters import to_bytes

with open('filename.txt', 'wb') as my_file:
    my_file.write(to_bytes(some_text_string))
```

Note that we don't have to catch `UnicodeError` here because we're transforming to UTF-8 and all text strings in Python can be transformed back to UTF-8.

Filesystem interaction

Dealing with filenames often involves dropping back to bytes because on UNIX-like systems filenames are bytes. On Python 2, if we pass a text string to these functions, the text string will be converted to a byte string inside of the function and a traceback will occur if non-ASCII characters are present. In Python 3, a traceback will only occur if the text string can't be decoded in the current locale, but it's still good to be explicit and have code which works on both versions:

```
import os.path

from ansible.module_utils.common.text.converters import to_bytes

filename = u'/var/tmp/.txt'
f = open(to_bytes(filename), 'wb')
mtime = os.path.getmtime(to_bytes(filename))
b_filename = os.path.expandvars(to_bytes(filename))
if os.path.exists(to_bytes(filename)):
    pass
```

When you are only manipulating a filename as a string without talking to the filesystem (or a C library which talks to the filesystem) you can often get away without converting to bytes:

```
import os.path

os.path.join(u'/var/tmp/café', u'')
os.path.split(u'/var/tmp/café/')
```

On the other hand, if the code needs to manipulate the filename and also talk to the filesystem, it can be more convenient to transform to bytes right away and manipulate in bytes.

Warning: Make sure all variables passed to a function are the same type. If you're working with something like `os.path.join()` which takes multiple strings and uses them in combination, you need to make sure that all the types are the same (either all bytes or all text). Mixing bytes and text will cause tracebacks.

Interacting with other programs

Interacting with other programs goes through the operating system and C libraries and operates on things that the UNIX kernel defines. These interfaces are all byte-oriented so the Python interface is byte oriented as well. On both Python 2 and Python 3, byte strings should be given to Python's subprocess library and byte strings should be expected back from it.

One of the main places in Ansible's controller code that we interact with other programs is the connection plugins' `exec_command` methods. These methods transform any text strings they receive in the command (and arguments to the command) to execute into bytes and return stdout and stderr as byte strings Higher level functions (like action plugins' `_low_level_execute_command`) transform the output into text strings.

Module string strategy: Native String

In modules we use a strategy known as Native Strings. This makes things easier on the community members who maintain so many of Ansible's modules, by not breaking backwards compatibility by mandating that all strings inside of modules are text and converting between text and bytes at the borders.

Native strings refer to the type that Python uses when you specify a bare string literal:

```
"This is a native string"
```

In Python 2, these are byte strings. In Python 3 these are text strings. Modules should be coded to expect bytes on Python 2 and text on Python 3.

Module_utils string strategy: hybrid

In `module_utils` code we use a hybrid string strategy. Although Ansible's `module_utils` code is largely like module code, some pieces of it are used by the controller as well. So it needs to be compatible with modules and with the controller's assumptions, particularly the string strategy. The module_utils code attempts to accept native strings as input to its functions and emit native strings as their output.

In `module_utils` code:

- Functions **must** accept string parameters as either text strings or byte strings.
- Functions may return either the same type of string as they were given or the native string type for the Python version they are run on.
- Functions that return strings **must** document whether they return strings of the same type as they were given or native strings.

Module-utils functions are therefore often very defensive in nature. They convert their string parameters into text (using `ansible.module_utils.common.text.converters.to_text`) at the beginning of the function, do their work, and then convert the return values into the native string type (using `ansible.module_utils.common.text.converters.to_native`) or back to the string type that their parameters received.

Tips, tricks, and idioms for Python 2/Python 3 compatibility

Use forward-compatibility boilerplate

Use the following boilerplate code at the top of all python files to make certain constructs act the same way on Python 2 and Python 3:

```
# Make coding more python3-ish
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type
```

`__metaclass__ = type` makes all classes defined in the file into new-style classes without explicitly inheriting from `object`.

The `__future__` imports do the following:

absolute_import
Makes imports look in `sys.path` for the modules being imported, skipping the directory in which the module doing the importing lives. If the code wants to use the directory in which the module doing the importing, there's a new dot notation to do so.

division
Makes division of integers always return a float. If you need to find the quotient use `x // y` instead of `x / y`.

print_function
Changes `print` from a keyword into a function.

See also:

- PEP 0328: Absolute Imports
- PEP 0238: Division
- PEP 3105: Print function

Prefix byte strings with `b_`

Since mixing text and bytes types leads to tracebacks we want to be clear about what variables hold text and what variables hold bytes. We do this by prefixing any variable holding bytes with `b_`. For instance:

```
filename = u'/var/tmp/café.txt'
b_filename = to_bytes(filename)
with open(b_filename) as f:
    data = f.read()
```

We do not prefix the text strings instead because we only operate on byte strings at the borders, so there are fewer variables that need bytes than text.

Import Ansible's bundled Python `six` library

The third-party Python six library exists to help projects create code that runs on both Python 2 and Python 3. Ansible includes a version of the library in module_utils so that other modules can use it without requiring that it is installed on the remote system. To make use of it, import it like this:

```
from ansible.module_utils import six
```

Note: Ansible can also use a system copy of six

Ansible will use a system copy of six if the system copy is a later version than the one Ansible bundles.

Handle exceptions with `as`

In order for code to function on Python 2.6+ and Python 3, use the new exception-catching syntax which uses the `as` keyword:

```
try:
    a = 2/0
except ValueError as e:
    module.fail_json(msg="Tried to divide by zero: %s" % e)
```

Do **not** use the following syntax as it will fail on every version of Python 3:

```
try:
    a = 2/0
except ValueError, e:
    module.fail_json(msg="Tried to divide by zero: %s" % e)
```

Update octal numbers

In Python 2.x, octal literals could be specified as `0755`. In Python 3, octals must be specified as `0o755`.

String formatting for controller code

Use `str.format()` for Python 2.6 compatibility

Starting in Python 2.6, strings gained a method called `format()` to put strings together. However, one commonly used feature of `format()` wasn't added until Python 2.7, so you need to remember not to use it in Ansible code:

```
# Does not work in Python 2.6!
new_string = "Dear {}, Welcome to {}".format(username, location)

# Use this instead
new_string = "Dear {0}, Welcome to {1}".format(username, location)
```

Both of the format strings above map positional arguments of the `format()` method into the string. However, the first version doesn't work in Python 2.6. Always remember to put numbers into the placeholders so the code is compatible with Python 2.6.

See also:

Python documentation on format strings:

- format strings in 2.6
- format strings in 3.x

Use percent format with byte strings

In Python 3.x, byte strings do not have a `format()` method. However, it does have support for the older, percent-formatting.

```
b_command_line = b'ansible-playbook --become-user %s -K %s' % (user, playbook_file)
```

Note: Percent formatting added in Python 3.5

Percent formatting of byte strings was added back into Python 3 in 3.5. This isn't a problem for us because Python 3.5 is our minimum version. However, if you happen to be testing Ansible code with Python 3.4 or earlier, you will find that the byte string formatting here won't work. Upgrade to Python 3.5 to test.

See also:

Python documentation on percent formatting

1.17.7 Debugging modules

Detailed debugging steps

Ansible modules are put together as a zip file consisting of the module file and the various Python module boilerplate inside of a wrapper script. To see what is actually happening in the module, you need to extract the file from the wrapper. The wrapper script provides helper methods that let you do that.

The following steps use `localhost` as the target host, but you can use the same steps to debug against remote hosts as well. For a simpler approach to debugging without using the temporary files, see *simple debugging*.

1. Set *ANSIBLE_KEEP_REMOTE_FILES* to 1 on the control host so Ansible will keep the remote module files instead of deleting them after the module finishes executing. Use the `-vvv` option to make Ansible more verbose. This will display the file name of the temporary module file.

   ```
   $ ANSIBLE_KEEP_REMOTE_FILES=1 ansible localhost -m ping -a 'data=debugging_
   ↪session' -vvv
   <127.0.0.1> ESTABLISH LOCAL CONNECTION FOR USER: badger
   <127.0.0.1> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo $HOME/.ansible/
   ↪tmp/ansible-tmp-1461434734.35-235318071810595 `" && echo "` echo $HOME/.
   ↪ansible/tmp/ansible-tmp-1461434734.35-235318071810595 `" )'
   <127.0.0.1> PUT /var/tmp/tmpjdbJ1w TO /home/badger/.ansible/tmp/ansible-
   ↪tmp-1461434734.35-235318071810595/AnsiballZ_ping.py
   <127.0.0.1> EXEC /bin/sh -c 'LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8 LC_
   ↪MESSAGES=en_US.UTF-8 /usr/bin/python /home/badger/.ansible/tmp/ansible-
   ↪tmp-1461434734.35-235318071810595/AnsiballZ_ping.py && sleep 0'
   localhost | SUCCESS => {
       "changed": false,
       "invocation": {
           "module_args": {
               "data": "debugging_session"
           },
           "module_name": "ping"
       },
       "ping": "debugging_session"
   }
   ```

2. Navigate to the temporary directory from the previous step. If the previous command was run against a remote host, connect to that host first before trying to navigate to the temporary directory.

```
$ ssh remotehost  # only if not debugging against localhost
$ cd /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595
```

3. Run the wrapper's `explode` command to turn the string into some Python files that you can work with.

   ```
   $ python AnsiballZ_ping.py explode
   Module expanded into:
   /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/debug_
   ↪dir
   ```

If you want to examine the wrapper file you can. It will show a small Python script with a large base64 encoded string. The string contains the module to execute.

4. When you look into the temporary directory you'll see a structure like this:

```
├── AnsiballZ_ping.py
└── debug_dir
    ├── ansible
    │   ├── __init__.py
    │   ├── module_utils
    │   │   ├── __init__.py
    │   │   ├── _text.py
    │   │   ├── basic.py
    │   │   ├── common
    │   │   ├── compat
    │   │   ├── distro
    │   │   ├── parsing
    │   │   ├── pycompat24.py
    │   │   └── six
    │   └── modules
    │       ├── __init__.py
    │       └── ping.py
    └── args
```

- `AnsiballZ_ping.py` is the Python script with the module code stored in a base64 encoded string. It contains various helper functions for executing the module.
- `ping.py` is the code for the module itself. You can modify this code to see what effect it would have on your module, or for debugging purposes.
- The `args` file contains a JSON string. The string is a dictionary containing the module arguments and other variables that Ansible passes into the module to change its behavior. Modify this file to change the parameters passed to the module.
- The `ansible` directory contains the module code in `modules` as well as code from `ansible.module_utils` that is used by the module. Ansible includes files for any `ansible.module_utils` imports in the module but not any files from any other module. If your module uses `ansible.module_utils.url` Ansible will include it for you. But if your module includes requests, then you'll have to make sure that the Python requests library is installed on the system before running the module.

You can modify files in this directory if you suspect that the module is having a problem in some of this boilerplate code rather than in the module code you have written.

5. Once you edit the code or arguments in the exploded tree, use the `execute` subcommand to run it:

```
$ python AnsiballZ_ping.py execute
{"invocation": {"module_args": {"data": "debugging_session"}}, "changed":␣
↪false, "ping": "debugging_session"}
```

This subcommand inserts the absolute path to `debug_dir` as the first item in `sys.path` and invokes the script using the arguments in the `args` file. You can continue to run the module like this until you understand the problem. Then you can copy the changes back into your real module file and test that the real module works by using the `ansible` or `ansible-playbook`.

Simple debugging

The easiest way to run a debugger in a module, either local or remote, is to use epdb. Add `import epdb; epdb.serve()` in the module code on the control node at the desired break point. To connect to the debugger, run `epdb.connect()`. See the epdb documentation for how to specify the `host` and `port`. If connecting to a remote node, make sure to use a port that is allowed by any firewall between the control node and the remote node.

This technique should work with any remote debugger, but we do not guarantee any particular remote debugging tool will work.

The q library is another very useful debugging tool.

Since `print()` statements do not work inside modules, raising an exception is a good approach if you just want to see some specific data. Put `raise Exception(some_value)` somewhere in the module and run it normally. Ansible will handle this exception, pass the message back to the control node, and display it.

1.17.8 Module format and documentation

If you want to contribute your module to most Ansible collections, you must write your module in Python and follow the standard format described below. (Unless you're writing a Windows module, in which case the *Windows guidelines* apply.) In addition to following this format, you should review our *submission checklist*, *programming tips*, and *strategy for maintaining Python 2 and Python 3 compatibility*, as well as information about *testing* before you open a pull request.

Every Ansible module written in Python must begin with seven standard sections in a particular order, followed by the code. The sections in order are:

Note: Why don't the imports go first?

Keen Python programmers may notice that contrary to PEP 8's advice we don't put `imports` at the top of the file. This is because the `DOCUMENTATION` through `RETURN` sections are not used by the module code itself; they are essentially extra docstrings for the file. The imports are placed after these special variables for the same reason as PEP 8 puts the imports after the introductory comments and docstrings. This keeps the active parts of the code together and the pieces which are purely informational apart. The decision to exclude E402 is based on readability (which is what PEP 8 is about). Documentation strings in a module are much more similar to module level docstrings, than code, and are never utilized by the module itself. Placing the imports below this documentation and closer to the code, consolidates and groups all related code in a congruent manner to improve readability, debugging and understanding.

Warning: Copy old modules with care!

Some older Ansible modules have `imports` at the bottom of the file, `Copyright` notices with the full GPL prefix, and/or `DOCUMENTATION` fields in the wrong order. These are legacy files that need updating - do not copy them into new modules. Over time we are updating and correcting older modules. Please follow the guidelines on this page!

Note: For non-Python modules you still create a `.py` file for documentation purposes. Starting at ansible-core 2.14 you can instead choose to create a `.yml` file that has the same data structure, but in pure YAML. With YAML files, the examples below are easy to use by removing Python quoting and substituting `=` for `:`, for example `DOCUMENTATION = r''' ... '''` ` to ``DOCUMENTATION: ...` and removing closing quotes. *Adjacent YAML documentation files*

Python shebang & UTF-8 coding

Begin your Ansible module with `#!/usr/bin/python` - this "shebang" allows `ansible_python_interpreter` to work. Follow the shebang immediately with `# -*- coding: utf-8 -*-` to clarify that the file is UTF-8 encoded.

Note: Using `#!/usr/bin/env`, makes `env` the interpreter and bypasses `ansible_<interpreter>_interpreter` logic.

Note: If you develop the module using a different scripting language, adjust the interpreter accordingly (`#!/usr/bin/<interpreter>`) so `ansible_<interpreter>_interpreter` can work for that specific language.

Note: Binary modules do not require a shebang or an interpreter.

Copyright and license

After the shebang and UTF-8 coding, add a copyright line with the original copyright holder and a license declaration. The license declaration should be ONLY one line, not the full GPL prefix.:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

# Copyright: Contributors to the Ansible project
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl-3.0.
↪txt)
```

Additions to the module (for instance, rewrites) are not permitted to add additional copyright lines other than the default copyright statement if missing:

```
# Copyright: Contributors to the Ansible project
```

Any legal review will include the source control history, so an exhaustive copyright header is not necessary. Please do not include a copyright year. If the existing copyright statement includes a year, do not edit the existing copyright year. Any existing copyright header should not be modified without permission from the copyright author.

ANSIBLE_METADATA block

Since we moved to collections we have deprecated the METADATA functionality, it is no longer required for modules, but it will not break anything if present.

DOCUMENTATION block

After the shebang, the UTF-8 coding, the copyright line, and the license section comes the `DOCUMENTATION` block. Ansible's online module documentation is generated from the `DOCUMENTATION` blocks in each module's source code. The `DOCUMENTATION` block must be valid YAML. You may find it easier to start writing your `DOCUMENTATION` string in an *editor with YAML syntax highlighting* before you include it in your Python file. You can start by copying our example documentation string into your module file and modifying it. If you run into syntax issues in your YAML, you can validate it on the YAML Lint website.

Module documentation should briefly and accurately define what each module and option does, and how it works with others in the underlying system. Documentation should be written for broad audience–readable both by experts and non-experts.

- Descriptions should always start with a capital letter and end with a full stop. Consistency always helps.
- Verify that arguments in doc and module spec dict are identical.
- For password / secret arguments `no_log=True` should be set.
- For arguments that seem to contain sensitive information but **do not** contain secrets, such as "password_length", set `no_log=False` to disable the warning message.
- If an option is only sometimes required, describe the conditions. For example, "Required when I(state=present)."
- If your module allows `check_mode`, reflect this fact in the documentation.

To create clear, concise, consistent, and useful documentation, follow the *style guide*.

Each documentation field is described below. Before committing your module documentation, please test it at the command line and as HTML:

- As long as your module file is *available locally*, you can use `ansible-doc -t module my_module_name` to view your module documentation at the command line. Any parsing errors will be obvious - you can view details by adding `-vvv` to the command.
- You should also test the HTML output of your module documentation.

Documentation fields

All fields in the `DOCUMENTATION` block are lower-case. All fields are required unless specified otherwise:

module

- The name of the module.
- Must be the same as the filename, without the `.py` extension.

short_description

- A short description which is displayed on the Collection Index page and `ansible-doc -l`.
- The `short_description` is displayed by `ansible-doc -l` without any category grouping, so it needs enough detail to explain the module's purpose without the context of the directory structure in which it lives.

- Unlike `description:`, `short_description` should not have a trailing period/full stop.

description

- A detailed description (generally two or more sentences).
- Must be written in full sentences, in other words, with capital letters and periods/full stops.
- Shouldn't mention the module name.
- Make use of multiple entries rather than using one long paragraph.
- Don't quote complete values unless it is required by YAML.

version_added

- The version of Ansible when the module was added.
- This is a string, and not a float, for example, `version_added: '2.1'`.
- In collections, this must be the collection version the module was added to, not the Ansible version. For example, `version_added: 1.0.0`.

author

- Name of the module author in the form `First Last (@GitHubID)`.
- Use a multi-line list if there is more than one author.
- Don't use quotes as it should not be required by YAML.

deprecated

- Marks modules that will be removed in future releases. See also *The lifecycle of an Ansible module or plugin*.

options

- Options are often called *parameters* or *arguments*. Because the documentation field is called *options*, we will use that term.
- If the module has no options (for example, it's a `_facts` module), all you need is one line: `options: {}`.
- If your module has options (in other words, accepts arguments), each option should be documented thoroughly. For each module option, include:

  option-name

  - Declarative operation (not CRUD), to focus on the final state, for example *online:*, rather than *is_online:*.
  - The name of the option should be consistent with the rest of the module, as well as other modules in the same category.
  - When in doubt, look for other modules to find option names that are used for the same purpose, we like to offer consistency to our users.

  description

  - Detailed explanation of what this option does. It should be written in full sentences.
  - The first entry is a description of the option itself; subsequent entries detail its use, dependencies, or format of possible values.
  - Should not list the possible values (that's what `choices:` is for, though it should explain what the values do if they aren't obvious).

- If an option is only sometimes required, describe the conditions. For example, "Required when I(state=present)."
- Mutually exclusive options must be documented as the final sentence on each of the options.

required

- Only needed if `true`.
- If missing, we assume the option is not required.

default

- If `required` is false/missing, `default` may be specified (assumed 'null' if missing).
- Ensure that the default value in the docs matches the default value in the code.
- The default field must not be listed as part of the description, unless it requires additional information or conditions.
- If the option is a boolean value, you can use any of the boolean values recognized by Ansible (such as `true`/`false` or `yes`/`no`). Document booleans as `true`/`false` for consistency and compatibility with ansible-lint.

choices

- List of option values.
- Should be absent if empty.

type

- Specifies the data type that option accepts, must match the `argspec`.
- If an argument is `type='bool'`, this field should be set to `type: bool` and no `choices` should be specified.
- If an argument is `type='list'`, `elements` should be specified.

elements

- Specifies the data type for list elements in case `type='list'`.

aliases

- List of optional name aliases.
- Generally not needed.

version_added

- Only needed if this option was extended after initial Ansible release, in other words, this is greater than the top level *version_added* field.
- This is a string, and not a float, for example, `version_added: '2.3'`.
- In collections, this must be the collection version the option was added to, not the Ansible version. For example, `version_added: 1.0.0`.

suboptions

- If this option takes a dict or list of dicts, you can define the structure here.
- See ansible_collections.azure.azcollection.azure_rm_securitygroup_module, ansible_collections.azure.azcollection.azure_rm_azurefirewall_module, and

ansible_collections.openstack.cloud.baremetal_node_action_module for examples.

requirements

- List of requirements (if applicable).
- Include minimum versions.

seealso

- A list of references to other modules, documentation or Internet resources
- In Ansible 2.10 and later, references to modules must use the FQCN or `ansible.builtin` for modules in `ansible-core`.
- A reference can be one of the following formats:

```
seealso:

# Reference by module name
- module: cisco.aci.aci_tenant

# Reference by module name, including description
- module: cisco.aci.aci_tenant
  description: ACI module to create tenants on a Cisco ACI fabric.

# Reference by rST documentation anchor
- ref: aci_guide
  description: Detailed information on how to manage your ACI␣
↪infrastructure using Ansible.

# Reference by rST documentation anchor (with custom title)
- ref: The official Ansible ACI guide <aci_guide>
  description: Detailed information on how to manage your ACI␣
↪infrastructure using Ansible.

# Reference by Internet resource
- name: APIC Management Information Model reference
  description: Complete reference of the APIC object model.
  link: https://developer.cisco.com/docs/apic-mim-ref/
```

- If you use `ref:` to link to an anchor that is not associated with a title, you must add a title to the ref for the link to work correctly.
- You can link to non-module plugins with `ref:` using the rST anchor, but plugin and module anchors are never associated with a title, so you must supply a title when you link to them. For example `ref: namespace.collection.plugin_name lookup plugin <ansible_collections.namespace.collection.plugin_name_lookup>`.

notes

- Details of any important information that doesn't fit in one of the above sections.
- For example, whether `check_mode` is or is not supported.

Linking and other format macros within module documentation

You can link from your module documentation to other module docs, other resources on docs.ansible.com, and resources elsewhere on the internet with the help of some pre-defined macros. The correct formats for these macros are:

- `L()` for links with a heading. For example: `See L(Ansible Automation Platform,https://www.ansible.com/products/automation-platform).` As of Ansible 2.10, do not use `L()` for relative links between Ansible documentation and collection documentation.
- `U()` for URLs. For example: `See U(https://www.ansible.com/products/automation-platform) for an overview.`
- `R()` for cross-references with a heading (added in Ansible 2.10). For example: `See R(Cisco IOS Platform Guide,ios_platform_options)`. Use the RST anchor for the cross-reference. See *Adding anchors* for details.
- `M()` for module names. For example: `See also M(ansible.builtin.yum) or M(community.general.apt_rpm)`.

There are also some macros which do not create links but we use them to display certain types of content in a uniform way:

- `I()` for option names. For example: `Required if I(state=present).` This is italicized in the documentation.
- `C()` for files, option values, and inline code. For example: `If not set the environment variable C(ACME_PASSWORD) will be used.` or `Use C(var | foo.bar.my_filter) to transform C(var) into the required format.` This displays with a mono-space font in the documentation.
- `B()` currently has no standardized usage. It is displayed in boldface in the documentation.
- `HORIZONTALLINE` is used sparingly as a separator in long descriptions. It becomes a horizontal rule (the <hr> html tag) in the documentation.

Note: For links between modules and documentation within a collection, you can use any of the options above. For links outside of your collection, use `R()` if available. Otherwise, use `U()` or `L()` with full URLs (not relative links). For modules, use `M()` with the FQCN or `ansible.builtin` as shown in the example. If you are creating your own documentation site, you will need to use the intersphinx extension to convert `R()` and `M()` to the correct links.

Note: To refer to a group of modules in a collection, use `R()`. When a collection is not the right granularity, use `C(..)`:

- `Refer to the R(kubernetes.core collection, plugins_in_kubernetes.core) for information on managing kubernetes clusters.`
- `The C(win_*) modules (spread across several collections) allow you to manage various aspects of windows hosts.`

Note: Because it stands out better, use `seealso` for general references over the use of notes or adding links to the description.

Documentation fragments

If you are writing multiple related modules, they may share common documentation, such as authentication details, file mode settings, `notes:` or `seealso:` entries. Rather than duplicate that information in each module's `DOCUMENTATION` block, you can save it once as a doc_fragment plugin and use it in each module's documentation. In Ansible, shared documentation fragments are contained in a `ModuleDocFragment` class in lib/ansible/plugins/doc_fragments/ or the equivalent directory in a collection. To include a documentation fragment, add `extends_documentation_fragment: FRAGMENT_NAME` in your module documentation. Use the fully qualified collection name for the FRAGMENT_NAME (for example, `kubernetes.core.k8s_auth_options`).

Modules should only use items from a doc fragment if the module will implement all of the interface documented there in a manner that behaves the same as the existing modules which import that fragment. The goal is that items imported from the doc fragment will behave identically when used in another module that imports the doc fragment.

By default, only the `DOCUMENTATION` property from a doc fragment is inserted into the module documentation. It is possible to define additional properties in the doc fragment in order to import only certain parts of a doc fragment or mix and match as appropriate. If a property is defined in both the doc fragment and the module, the module value overrides the doc fragment.

Here is an example doc fragment named `example_fragment.py`:

```
class ModuleDocFragment(object):
    # Standard documentation
    DOCUMENTATION = r'''
    options:
      # options here
    '''

    # Additional section
    OTHER = r'''
    options:
      # other options here
    '''
```

To insert the contents of `OTHER` in a module:

```
extends_documentation_fragment: example_fragment.other
```

Or use both :

```
extends_documentation_fragment:
  - example_fragment
  - example_fragment.other
```

New in version 2.8.

Since Ansible 2.8, you can have user-supplied doc_fragments by using a `doc_fragments` directory adjacent to play or role, just like any other plugin.

For example, all AWS modules should include:

```
extends_documentation_fragment:
- aws
- ec2
```

Using documentation fragments in collections describes how to incorporate documentation fragments in a collection.

EXAMPLES block

After the shebang, the UTF-8 coding, the copyright line, the license section, and the `DOCUMENTATION` block comes the `EXAMPLES` block. Here you show users how your module works with real-world examples in multi-line plain-text YAML format. The best examples are ready for the user to copy and paste into a playbook. Review and update your examples with every change to your module.

Per playbook best practices, each example should include a `name:` line:

```
EXAMPLES = r'''
- name: Ensure foo is installed
  namespace.collection.modulename:
    name: foo
    state: present
'''
```

The `name:` line should be capitalized and not include a trailing dot.

Use a fully qualified collection name (FQCN) as a part of the module's name like in the example above. For modules in `ansible-core`, use the `ansible.builtin.` identifier, for example `ansible.builtin.debug`.

If your examples use boolean options, use yes/no values. Since the documentation generates boolean values as yes/no, having the examples use these values as well makes the module documentation more consistent.

If your module returns facts that are often needed, an example of how to use them can be helpful.

RETURN block

After the shebang, the UTF-8 coding, the copyright line, the license section, `DOCUMENTATION` and `EXAMPLES` blocks comes the `RETURN` block. This section documents the information the module returns for use by other modules.

If your module doesn't return anything (apart from the standard returns), this section of your module should read: `RETURN = r''' # '''` Otherwise, for each value returned, provide the following fields. All fields are required unless specified otherwise.

return name
: Name of the returned field.

  description
  : Detailed description of what this value represents. Capitalized and with trailing dot.

  returned
  : When this value is returned, such as `always`, `changed` or `success`. This is a string and can contain any human-readable content.

  type
  : Data type.

  elements
  : If `type='list'`, specifies the data type of the list's elements.

  sample
  : One or more examples.

  version_added
  : Only needed if this return was extended after initial Ansible release, in other words, this is greater than the top level *version_added* field. This is a string, and not a float, for example, `version_added: '2.3'`.

contains
Optional. To describe nested return values, set `type: dict`, or `type: list/elements: dict`, or if you really have to, `type: complex`, and repeat the elements above for each sub-field.

Here are two example `RETURN` sections, one with three simple fields and one with a complex nested field:

```
RETURN = r'''
dest:
    description: Destination file/path.
    returned: success
    type: str
    sample: /path/to/file.txt
src:
    description: Source file used for the copy on the target machine.
    returned: changed
    type: str
    sample: /home/httpd/.ansible/tmp/ansible-tmp-1423796390.97-147729857856000/source
md5sum:
    description: MD5 checksum of the file after running copy.
    returned: when supported
    type: str
    sample: 2a5aeecc61dc98c4d780b14b330e3282
'''

RETURN = r'''
packages:
    description: Information about package requirements.
    returned: success
    type: dict
    contains:
        missing:
            description: Packages that are missing from the system.
            returned: success
            type: list
            elements: str
            sample:
                - libmysqlclient-dev
                - libxml2-dev
        badversion:
            description: Packages that are installed but at bad versions.
            returned: success
            type: list
            elements: dict
            sample:
                - package: libxml2-dev
                  version: 2.9.4+dfsg1-2
                  constraint: ">= 3.0"
'''
```

Python imports

After the shebang, the UTF-8 coding, the copyright line, the license, and the sections for `DOCUMENTATION`, `EXAMPLES`, and `RETURN`, you can finally add the python imports. All modules must use Python imports in the form:

```
from module_utils.basic import AnsibleModule
```

The use of "wildcard" imports such as `from module_utils.basic import *` is no longer allowed.

Testing module documentation

To test Ansible documentation locally please follow instruction. To test documentation in collections, please see *Build a docsite with antsibull-docs*.

1.17.9 Adjacent YAML documentation files

YAML documentation for plugins

For most Ansible plugins, the documentation is in the same file as the code. This approach does not work for cases when:

- Multiple plugins are defined in the same file, such as tests and filters.
- Plugins are written in a language other than Python (modules).

These cases require plugins to provide documentation in an adjacent `.py` file. As of ansible-core 2.14, you can provide documentation as adjacent YAML files instead. The format of a YAML documentation file is nearly identical to its Python equivalent, except it is pure YAML.

YAML format

In Python each section is a variable `DOCUMENTATION = r""" ... """` while in YAML it is a mapping key `DOCUMENTATION: ...`.

Here is a longer example that shows documentation as embedded in a Python file:

```
DOCUMENTATION = r'''
  description: something
  options:
    option_name:
      description: describe this config option
      default: default value for this config option
      env:
        - name: NAME_OF_ENV_VAR
      ini:
        - section: section_of_ansible.cfg_where_this_config_option_is_defined
```

(continues on next page)

(continued from previous page)

```
          key: key_used_in_ansible.cfg
      vars:
        - name: name_of_ansible_var
        - name: name_of_second_var
          version_added: X.x
      required: True/False
      type: boolean/float/integer/list/none/path/pathlist/pathspec/string/tmppath
      version_added: X.x
'''

EXAMPLES = r'''
  # TODO: write examples
'''
```

This example shows the same documentation in YAML format:

```
DOCUMENTATION:
  description: something
  options:
    option_name:
      description: describe this config option
      default: default value for this config option
      env:
        - name: NAME_OF_ENV_VAR
      ini:
        - section: section_of_ansible.cfg_where_this_config_option_is_defined
          key: key_used_in_ansible.cfg
      vars:
        - name: name_of_ansible_var
        - name: name_of_second_var
          version_added: X.x
      required: True/False
      type: boolean/float/integer/list/none/path/pathlist/pathspec/string/tmppath
      version_added: X.x

EXAMPLES: # TODO: write examples
```

As the examples above show, Python variables already contain YAML. The main change to use YAML documentation is to simply move the YAML out of such variables.

> Any adjacent YAML documentation files must be in the same directory as the plugin or module that they document. This means the documentation is available in any directory that contains the plugins or modules.

Supported plugin types

YAML documentation is mainly intended for filters, tests and modules. While it is possible to use with other plugin types, Ansible always recommends having documentation in the same file as the code for most cases.

See also:

Collection Index
Browse existing collections, modules, and plugins

Python API
Learn about the Python API for task execution

Developing dynamic inventory
Learn about how to develop dynamic inventory sources

Developing modules
Learn about how to write Ansible modules

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

1.17.10 Windows module development walkthrough

In this section, we will walk through developing, testing, and debugging an Ansible Windows module.

Because Windows modules are written in Powershell and need to be run on a Windows host, this guide differs from the usual development walkthrough guide.

What's covered in this section:

Windows environment setup

Unlike Python module development which can be run on the host that runs Ansible, Windows modules need to be written and tested for Windows hosts. While evaluation editions of Windows can be downloaded from Microsoft, these images are usually not ready to be used by Ansible without further modification. The easiest way to set up a Windows host so that it is ready to by used by Ansible is to set up a virtual machine using Vagrant. Vagrant can be used to download existing OS images called *boxes* that are then deployed to a hypervisor like VirtualBox. These boxes can either be created and stored offline or they can be downloaded from a central repository called Vagrant Cloud.

This guide will use the Vagrant boxes created by the packer-windoze repository which have also been uploaded to Vagrant Cloud. To find out more info on how these images are created, please go to the GitHub repo and look at the `README` file.

Before you can get started, the following programs must be installed (please consult the Vagrant and VirtualBox documentation for installation instructions):

- Vagrant
- VirtualBox

Create a Windows server in a VM

To create a single Windows Server 2016 instance, run the following:

```
vagrant init jborean93/WindowsServer2016
vagrant up
```

This will download the Vagrant box from Vagrant Cloud and add it to the local boxes on your host and then start up that instance in VirtualBox. When starting for the first time, the Windows VM will run through the sysprep process and then create a HTTP and HTTPS WinRM listener automatically. Vagrant will finish its process once the listeners are online, after which the VM can be used by Ansible.

Create an Ansible inventory

The following Ansible inventory file can be used to connect to the newly created Windows VM:

```
[windows]
WindowsServer  ansible_host=127.0.0.1

[windows:vars]
ansible_user=vagrant
ansible_password=vagrant
ansible_port=55986
ansible_connection=winrm
ansible_winrm_transport=ntlm
ansible_winrm_server_cert_validation=ignore
```

Note: The port `55986` is automatically forwarded by Vagrant to the Windows host that was created, if this conflicts with an existing local port then Vagrant will automatically use another one at random and display show that in the output.

The OS that is created is based on the image set. The following images can be used:

- jborean93/WindowsServer2012

- jborean93/WindowsServer2012R2
- jborean93/WindowsServer2016
- jborean93/WindowsServer2019
- jborean93/WindowsServer2022

When the host is online, it can accessible by RDP on `127.0.0.1:3389` but the port may differ depending if there was a conflict. To get rid of the host, run `vagrant destroy --force` and Vagrant will automatically remove the VM and any other files associated with that VM.

While this is useful when testing modules on a single Windows instance, these host won't work without modification with domain based modules. The Vagrantfile at ansible-windows can be used to create a test domain environment to be used in Ansible. This repo contains three files which are used by both Ansible and Vagrant to create multiple Windows hosts in a domain environment. These files are:

- `Vagrantfile`: The Vagrant file that reads the inventory setup of `inventory.yml` and provisions the hosts that are required
- `inventory.yml`: Contains the hosts that are required and other connection information such as IP addresses and forwarded ports
- `main.yml`: Ansible playbook called by Vagrant to provision the domain controller and join the child hosts to the domain

By default, these files will create the following environment:

- A single domain controller running on Windows Server 2016
- Five child hosts for each major Windows Server version joined to that domain
- A domain with the DNS name `domain.local`
- A local administrator account on each host with the username `vagrant` and password `vagrant`
- A domain admin account `vagrant-domain@domain.local` with the password `VagrantPass1`

The domain name and accounts can be modified by changing the variables `domain_*` in the `inventory.yml` file if it is required. The inventory file can also be modified to provision more or less servers by changing the hosts that are defined under the `domain_children` key. The host variable `ansible_host` is the private IP that will be assigned to the VirtualBox host only network adapter while `vagrant_box` is the box that will be used to create the VM.

Provisioning the environment

To provision the environment as is, run the following:

```
git clone https://github.com/jborean93/ansible-windows.git
cd vagrant
vagrant up
```

Note: Vagrant provisions each host sequentially so this can take some time to complete. If any errors occur during the Ansible phase of setting up the domain, run `vagrant provision` to rerun just that step.

Unlike setting up a single Windows instance with Vagrant, these hosts can also be accessed using the IP address directly as well as through the forwarded ports. It is easier to access it over the host only network adapter as the normal protocol ports are used, for example RDP is still over `3389`. In cases where the host cannot be resolved using the host only network IP, the following protocols can be access over `127.0.0.1` using these forwarded ports:

- RDP: 295xx

- `SSH`: 296xx
- `WinRM HTTP`: 297xx
- `WinRM HTTPS`: 298xx
- `SMB`: 299xx

Replace `xx` with the entry number in the inventory file where the domain controller started with `00` and is incremented from there. For example, in the default `inventory.yml` file, WinRM over HTTPS for `SERVER2012R2` is forwarded over port `29804` as it's the fourth entry in `domain_children`.

Windows new module development

When creating a new module there are a few things to keep in mind:

- Module code is in Powershell (.ps1) files while the documentation is contained in Python (.py) files of the same name
- Avoid using `Write-Host/Debug/Verbose/Error` in the module and add what needs to be returned to the `$module.Result` variable
- To fail a module, call `$module.FailJson("failure message here")`, an Exception or ErrorRecord can be set to the second argument for a more descriptive error message
- You can pass in the exception or ErrorRecord as a second argument to `FailJson("failure", $_)` to get a more detailed output
- Most new modules require check mode and integration tests before they are merged into the main Ansible codebase
- Avoid using try/catch statements over a large code block, rather use them for individual calls so the error message can be more descriptive
- Try and catch specific exceptions when using try/catch statements
- Avoid using PSCustomObjects unless necessary
- Look for common functions in `./lib/ansible/module_utils/powershell/` and use the code there instead of duplicating work. These can be imported by adding the line `#Requires -Module *` where * is the filename to import, and will be automatically included with the module code sent to the Windows target when run through Ansible
- As well as PowerShell module utils, C# module utils are stored in `./lib/ansible/module_utils/csharp/` and are automatically imported in a module execution if the line `#AnsibleRequires -CSharpUtil *` is present
- C# and PowerShell module utils achieve the same goal but C# allows a developer to implement low level tasks, such as calling the Win32 API, and can be faster in some cases
- Ensure the code runs under Powershell v3 and higher on Windows Server 2012 and higher; if higher minimum Powershell or OS versions are required, ensure the documentation reflects this clearly
- Ansible runs modules under strictmode version 2.0. Be sure to test with that enabled by putting `Set-StrictMode -Version 2.0` at the top of your dev script
- Favor native Powershell cmdlets over executable calls if possible
- Use the full cmdlet name instead of aliases, for example `Remove-Item` over `rm`
- Use named parameters with cmdlets, for example `Remove-Item -Path C:\temp` over `Remove-Item C:\temp`

A very basic Powershell module win_environment incorporates best practices for Powershell modules. It demonstrates how to implement check-mode and diff-support, and also shows a warning to the user when a specific condition is met.

A slightly more advanced module is win_uri which additionally shows how to use different parameter types (bool, str, int, list, dict, path) and a selection of choices for parameters, how to fail a module and how to handle exceptions.

As part of the new `AnsibleModule` wrapper, the input parameters are defined and validated based on an argument spec. The following options can be set at the root level of the argument spec:

- `mutually_exclusive`: A list of lists, where the inner list contains module options that cannot be set together
- `no_log`: Stops the module from emitting any logs to the Windows Event log
- `options`: A dictionary where the key is the module option and the value is the spec for that option
- `required_by`: A dictionary where the option(s) specified by the value must be set if the option specified by the key is also set
- **`required_if`: A list of lists where the inner list contains 3 or 4 elements;**
    - The first element is the module option to check the value against
    - The second element is the value of the option specified by the first element, if matched then the required if check is run
    - The third element is a list of required module options when the above is matched
    - An optional fourth element is a boolean that states whether all module options in the third elements are required (default: `$false`) or only one (`$true`)
- `required_one_of`: A list of lists, where the inner list contains module options where at least one must be set
- `required_together`: A list of lists, where the inner list contains module options that must be set together
- `supports_check_mode`: Whether the module supports check mode, by default this is `$false`

The actual input options for a module are set within the `options` value as a dictionary. The keys of this dictionary are the module option names while the values are the spec of that module option. Each spec can have the following options set:

- `aliases`: A list of aliases for the module option
- `choices`: A list of valid values for the module option, if `type=list` then each list value is validated against the choices and not the list itself
- `default`: The default value for the module option if not set
- `deprecated_aliases`: A list of hashtables that define aliases that are deprecated and the versions they will be removed in. Each entry must contain the keys `name` and `collection_name` with either `version` or `date`
- `elements`: When `type=list`, this sets the type of each list value, the values are the same as `type`
- `no_log`: Will sanitise the input value before being returned in the `module_invocation` return value
- `removed_in_version`: States when a deprecated module option is to be removed, a warning is displayed to the end user if set
- `removed_at_date`: States the date (YYYY-MM-DD) when a deprecated module option will be removed, a warning is displayed to the end user if set
- `removed_from_collection`: States from which collection the deprecated module option will be removed; must be specified if one of `removed_in_version` and `removed_at_date` is specified
- `required`: Will fail when the module option is not set
- **`type`: The type of the module option, if not set then it defaults to `str`. The valid types are;**

  - `bool`: A boolean value
  - `dict`: A dictionary value, if the input is a JSON or key=value string then it is converted to dictionary
  - `float`: A float or Single value
  - `int`: An Int32 value
  - `json`: A string where the value is converted to a JSON string if the input is a dictionary
  - `list`: A list of values, `elements=<type>` can convert the individual list value types if set. If `elements=dict` then `options` is defined, the values will be validated against the argument spec. When the input is a string then the string is split by `,` and any whitespace is trimmed
  - `path`: A string where values likes `%TEMP%` are expanded based on environment values. If the input value starts with `\\?\` then no expansion is run
  - `raw`: No conversions occur on the value passed in by Ansible
  - `sid`: Will convert Windows security identifier values or Windows account names to a SecurityIdentifier value
  - `str`: The value is converted to a string

When `type=dict`, or `type=list` and `elements=dict`, the following keys can also be set for that module option:

- `apply_defaults`: The value is based on the `options` spec defaults for that key if `True` and null if `False`. Only valid when the module option is not defined by the user and `type=dict`.
- `mutually_exclusive`: Same as the root level `mutually_exclusive` but validated against the values in the sub dict
- `options`: Same as the root level `options` but contains the valid options for the sub option
- `required_if`: Same as the root level `required_if` but validated against the values in the sub dict
- `required_by`: Same as the root level `required_by` but validated against the values in the sub dict
- `required_together`: Same as the root level `required_together` but validated against the values in the sub dict
- `required_one_of`: Same as the root level `required_one_of` but validated against the values in the sub dict

A module type can also be a delegate function that converts the value to whatever is required by the module option. For example the following snippet shows how to create a custom type that creates a `UInt64` value:

```
$spec = @{
    uint64_type = @{ type = [Func[[Object], [UInt64]]]{ [System.UInt64]::Parse($args[0])␣
↪} }
}
$uint64_type = $module.Params.uint64_type
```

When in doubt, look at some of the other core modules and see how things have been implemented there.

Sometimes there are multiple ways that Windows offers to complete a task; this is the order to favor when writing modules:

- Native Powershell cmdlets like `Remove-Item -Path C:\temp -Recurse`
- .NET classes like `[System.IO.Path]::GetRandomFileName()`
- WMI objects through the `New-CimInstance` cmdlet
- COM objects through `New-Object -ComObject` cmdlet
- Calls to native executables like `Secedit.exe`

PowerShell modules support a small subset of the #Requires options built into PowerShell as well as some Ansible-specific requirements specified by #AnsibleRequires. These statements can be placed at any point in the script, but are most commonly near the top. They are used to make it easier to state the requirements of the module without writing any of the checks. Each requires statement must be on its own line, but there can be multiple requires statements in one script.

These are the checks that can be used within Ansible modules:

- #Requires -Module Ansible.ModuleUtils.<module_util>: Added in Ansible 2.4, specifies a module_util to load in for the module execution.
- #Requires -Version x.y: Added in Ansible 2.5, specifies the version of PowerShell that is required by the module. The module will fail if this requirement is not met.
- #AnsibleRequires -PowerShell <module_util>: Added in Ansible 2.8, like #Requires -Module, this specifies a module_util to load in for module execution.
- #AnsibleRequires -CSharpUtil <module_util>: Added in Ansible 2.8, specifies a C# module_util to load in for the module execution.
- #AnsibleRequires -OSVersion x.y: Added in Ansible 2.5, specifies the OS build version that is required by the module and will fail if this requirement is not met. The actual OS version is derived from [Environment]::OSVersion.Version.
- #AnsibleRequires -Become: Added in Ansible 2.5, forces the exec runner to run the module with become, which is primarily used to bypass WinRM restrictions. If ansible_become_user is not specified then the SYSTEM account is used instead.

The #AnsibleRequires -PowerShell and #AnsibleRequires -CSharpUtil support further features such as:

- Importing a util contained in a collection (added in Ansible 2.9)
- Importing a util by relative names (added in Ansible 2.10)
- Specifying the util is optional by adding *-Optional* to the import declaration (added in Ansible 2.12).

See the below examples for more details:

```
# Imports the PowerShell Ansible.ModuleUtils.Legacy provided by Ansible itself
#AnsibleRequires -PowerShell Ansible.ModuleUtils.Legacy

# Imports the PowerShell my_util in the my_namesapce.my_name collection
#AnsibleRequires -PowerShell ansible_collections.my_namespace.my_name.plugins.module_
↪utils.my_util

# Imports the PowerShell my_util that exists in the same collection as the current module
#AnsibleRequires -PowerShell ..module_utils.my_util

# Imports the PowerShell Ansible.ModuleUtils.Optional provided by Ansible if it exists.
# If it does not exist then it will do nothing.
#AnsibleRequires -PowerShell Ansible.ModuleUtils.Optional -Optional

# Imports the C# Ansible.Process provided by Ansible itself
#AnsibleRequires -CSharpUtil Ansible.Process

# Imports the C# my_util in the my_namespace.my_name collection
#AnsibleRequires -CSharpUtil ansible_collections.my_namespace.my_name.plugins.module_
↪utils.my_util
```

(continues on next page)

(continued from previous page)

```
# Imports the C# my_util that exists in the same collection as the current module
#AnsibleRequires -CSharpUtil ..module_utils.my_util

# Imports the C# Ansible.Optional provided by Ansible if it exists.
# If it does not exist then it will do nothing.
#AnsibleRequires -CSharpUtil Ansible.Optional -Optional
```

For optional require statements, it is up to the module code to then verify whether the util has been imported before trying to use it. This can be done by checking if a function or type provided by the util exists or not.

While both `#Requires -Module` and `#AnsibleRequires -PowerShell` can be used to load a PowerShell module it is recommended to use `#AnsibleRequires`. This is because `#AnsibleRequires` supports collection module utils, imports by relative util names, and optional util imports.

C# module utils can reference other C# utils by adding the line `using Ansible.<module_util>;` to the top of the script with all the other using statements.

Windows module utilities

Like Python modules, PowerShell modules also provide a number of module utilities that provide helper functions within PowerShell. These module_utils can be imported by adding the following line to a PowerShell module:

```
#Requires -Module Ansible.ModuleUtils.Legacy
```

This will import the module_util at `./lib/ansible/module_utils/powershell/Ansible.ModuleUtils.Legacy.psm1` and enable calling all of its functions. As of Ansible 2.8, Windows module utils can also be written in C# and stored at `lib/ansible/module_utils/csharp`. These module_utils can be imported by adding the following line to a PowerShell module:

```
#AnsibleRequires -CSharpUtil Ansible.Basic
```

This will import the module_util at `./lib/ansible/module_utils/csharp/Ansible.Basic.cs` and automatically load the types in the executing process. C# module utils can reference each other and be loaded together by adding the following line to the using statements at the top of the util:

```
using Ansible.Become;
```

There are special comments that can be set in a C# file for controlling the compilation parameters. The following comments can be added to the script;

- `//AssemblyReference -Name <assembly dll> [-CLR [Core|Framework]]`: The assembly DLL to reference during compilation, the optional `-CLR` flag can also be used to state whether to reference when running under .NET Core, Framework, or both (if omitted)
- `//NoWarn -Name <error id> [-CLR [Core|Framework]]`: A compiler warning ID to ignore when compiling the code, the optional `-CLR` works the same as above. A list of warnings can be found at Compiler errors

As well as this, the following pre-processor symbols are defined;

- `CORECLR`: This symbol is present when PowerShell is running through .NET Core
- `WINDOWS`: This symbol is present when PowerShell is running on Windows
- `UNIX`: This symbol is present when PowerShell is running on Unix

A combination of these flags help to make a module util interoperable on both .NET Framework and .NET Core, here is an example of them in action:

```
#if CORECLR
using Newtonsoft.Json;
#else
using System.Web.Script.Serialization;
#endif

//AssemblyReference -Name Newtonsoft.Json.dll -CLR Core
//AssemblyReference -Name System.Web.Extensions.dll -CLR Framework

// Ignore error CS1702 for all .NET types
//NoWarn -Name CS1702

// Ignore error CS1956 only for .NET Framework
//NoWarn -Name CS1956 -CLR Framework
```

The following is a list of module_utils that are packaged with Ansible and a general description of what they do:

- ArgvParser: Utility used to convert a list of arguments to an escaped string compliant with the Windows argument parsing rules.
- CamelConversion: Utility used to convert camelCase strings/lists/dicts to snake_case.
- CommandUtil: Utility used to execute a Windows process and return the stdout/stderr and rc as separate objects.
- FileUtil: Utility that expands on the `Get-ChildItem` and `Test-Path` to work with special files like `C:\pagefile.sys`.
- Legacy: General definitions and helper utilities for Ansible module.
- LinkUtil: Utility to create, remove, and get information about symbolic links, junction points and hard inks.
- SID: Utilities used to convert a user or group to a Windows SID and vice versa.

For more details on any specific module utility and their requirements, please see the Ansible module utilities source code.

PowerShell module utilities can be stored outside of the standard Ansible distribution for use with custom modules. Custom module_utils are placed in a folder called `module_utils` located in the root folder of the playbook or role directory.

C# module utilities can also be stored outside of the standard Ansible distribution for use with custom modules. Like PowerShell utils, these are stored in a folder called `module_utils` and the filename must end in the extension `.cs`, start with `Ansible.` and be named after the namespace defined in the util.

The below example is a role structure that contains two PowerShell custom module_utils called `Ansible.ModuleUtils.ModuleUtil1`, `Ansible.ModuleUtils.ModuleUtil2`, and a C# util containing the namespace `Ansible.CustomUtil`:

```
meta/
  main.yml
defaults/
  main.yml
module_utils/
  Ansible.ModuleUtils.ModuleUtil1.psm1
  Ansible.ModuleUtils.ModuleUtil2.psm1
  Ansible.CustomUtil.cs
tasks/
  main.yml
```

Each PowerShell module_util must contain at least one function that has been exported with `Export-ModuleMember` at the end of the file. For example

```
Export-ModuleMember -Function Invoke-CustomUtil, Get-CustomInfo
```

Exposing shared module options

PowerShell module utils can easily expose common module options that a module can use when building its argument spec. This allows common features to be stored and maintained in one location and have those features used by multiple modules with minimal effort. Any new features or bugfixes added to one of these utils are then automatically used by the various modules that call that util.

An example of this would be to have a module util that handles authentication and communication against an API This util can be used by multiple modules to expose a common set of module options like the API endpoint, username, password, timeout, cert validation, and so on without having to add those options to each module spec.

The standard convention for a module util that has a shared argument spec would have

- **A `Get-<namespace.name.util name>Spec` function that outputs the common spec for a module**
  - It is highly recommended to make this function name be unique to the module to avoid any conflicts with other utils that can be loaded
  - The format of the output spec is a Hashtable in the same format as the `$spec` used for normal modules
- A function that takes in an `AnsibleModule` object called under the `-Module` parameter which it can use to get the shared options

Because these options can be shared across various module it is highly recommended to keep the module option names and aliases in the shared spec as specific as they can be. For example do not have a util option called `password`, rather you should prefix it with a unique name like `acme_password`.

Warning: Failure to have a unique option name or alias can prevent the util being used by module that also use those names or aliases for its own options.

The following is an example module util called `ServiceAuth.psm1` in a collection that implements a common way for modules to authentication with a service.

```
Invoke-MyServiceResource {
    [CmdletBinding()]
    param (
        [Parameter(Mandatory=$true)]
        [ValidateScript({ $_.GetType().FullName -eq 'Ansible.Basic.AnsibleModule' })]
        $Module,

        [Parameter(Mandatory=$true)]
        [String]
        $ResourceId,

        [String]
        $State = 'present'
    )
```

(continues on next page)

(continued from previous page)

```
    # Process the common module options known to the util
    $params = @{
        ServerUri = $Module.Params.my_service_url
    }
    if ($Module.Params.my_service_username) {
        $params.Credential = Get-MyServiceCredential
    }

    if ($State -eq 'absent') {
        Remove-MyService @params -ResourceId $ResourceId
    } else {
        New-MyService @params -ResourceId $ResourceId
    }
}

Get-MyNamespaceMyCollectionServiceAuthSpec {
    # Output the util spec
    @{
        options = @{
            my_service_url = @{ type = 'str'; required = $true }
            my_service_username = @{ type = 'str' }
            my_service_password = @{ type = 'str'; no_log = $true }
        }

        required_together = @(
            ,@('my_service_username', 'my_service_password')
        )
    }
}

$exportMembers = @{
    Function = 'Get-MyNamespaceMyCollectionServiceAuthSpec', 'Invoke-MyServiceResource'
}
Export-ModuleMember @exportMembers
```

For a module to take advantage of this common argument spec it can be set out like

```
#!powershell

# Include the module util ServiceAuth.psm1 from the my_namespace.my_collection collection
#AnsibleRequires -PowerShell ansible_collections.my_namespace.my_collection.plugins.
↪module_utils.ServiceAuth

# Create the module spec like normal
$spec = @{
    options = @{
        resource_id = @{ type = 'str'; required = $true }
        state = @{ type = 'str'; choices = 'absent', 'present' }
    }
}

# Create the module from the module spec but also include the util spec to merge into␣
```

(continues on next page)

(continued from previous page)

```
↪our own.
$module = [Ansible.Basic.AnsibleModule]::Create($args, $spec, @(Get-
↪MyNamespaceMyCollectionServiceAuthSpec))

# Call the ServiceAuth module util and pass in the module object so it can access the␣
↪module options.
Invoke-MyServiceResource -Module $module -ResourceId $module.Params.resource_id -State
↪$module.params.state

$module.ExitJson()
```

Note: Options defined in the module spec will always have precedence over a util spec. Any list values under the same key in a util spec will be appended to the module spec for that same key. Dictionary values will add any keys that are missing from the module spec and merge any values that are lists or dictionaries. This is similar to how the doc fragment plugins work when extending module documentation.

To document these shared util options for a module, create a doc fragment plugin that documents the options implemented by the module util and extend the module docs for every module that implements the util to include that fragment in its docs.

Windows playbook module testing

You can test a module with an Ansible playbook. For example:

- Create a playbook in any directory `touch testmodule.yml`.
- Create an inventory file in the same directory `touch hosts`.
- Populate the inventory file with the variables required to connect to a Windows host(s).
- Add the following to the new playbook file:

```
---
- name: test out windows module
  hosts: windows
  tasks:
  - name: test out module
    win_module:
      name: test name
```

- Run the playbook `ansible-playbook -i hosts testmodule.yml`

This can be useful for seeing how Ansible runs with the new module end to end. Other possible ways to test the module are shown below.

Windows debugging

Debugging a module currently can only be done on a Windows host. This can be useful when developing a new module or implementing bug fixes. These are some steps that need to be followed to set this up:

- Copy the module script to the Windows server
- Copy the folders `./lib/ansible/module_utils/powershell` and `./lib/ansible/module_utils/csharp` to the same directory as the script above
- Add an extra `#` to the start of any `#Requires -Module` lines in the module code, this is only required for any lines starting with `#Requires -Module`
- Add the following to the start of the module script that was copied to the server:

```
# Set $ErrorActionPreference to what's set during Ansible execution
$ErrorActionPreference = "Stop"

# Set the first argument as the path to a JSON file that contains the module args
$args = @("$($pwd.Path)\args.json")

# Or instead of an args file, set $complex_args to the pre-processed module args
$complex_args = @{
    _ansible_check_mode = $false
    _ansible_diff = $false
    path = "C:\temp"
    state = "present"
}

# Import any C# utils referenced with '#AnsibleRequires -CSharpUtil' or 'using Ansible.;
# The $_csharp_utils entries should be the context of the C# util files and not the path
Import-Module -Name "$($pwd.Path)\powershell\Ansible.ModuleUtils.AddType.psm1"
$_csharp_utils = @(
    [System.IO.File]::ReadAllText("$($pwd.Path)\csharp\Ansible.Basic.cs")
)
Add-CSharpType -References $_csharp_utils -IncludeDebugInfo

# Import any PowerShell modules referenced with '#Requires -Module`
Import-Module -Name "$($pwd.Path)\powershell\Ansible.ModuleUtils.Legacy.psm1"

# End of the setup code and start of the module code
#!powershell
```

You can add more args to `$complex_args` as required by the module or define the module options through a JSON file with the structure:

```
{
    "ANSIBLE_MODULE_ARGS": {
        "_ansible_check_mode": false,
        "_ansible_diff": false,
        "path": "C:\\temp",
        "state": "present"
    }
}
```

There are multiple IDEs that can be used to debug a Powershell script, two of the most popular ones are

- Powershell ISE
- Visual Studio Code

To be able to view the arguments as passed by Ansible to the module follow these steps.

- Prefix the Ansible command with *ANSIBLE_KEEP_REMOTE_FILES=1* to specify that Ansible should keep the exec files on the server.
- Log onto the Windows server using the same user account that Ansible used to execute the module.
- Navigate to `%TEMP%\..`. It should contain a folder starting with `ansible-tmp-`.
- Inside this folder, open the PowerShell script for the module.
- In this script is a raw JSON script under `$json_raw` which contains the module arguments under `module_args`. These args can be assigned manually to the `$complex_args` variable that is defined on your debug script or put in the `args.json` file.

Windows unit testing

Currently there is no mechanism to run unit tests for Powershell modules under Ansible CI.

Windows integration testing

Integration tests for Ansible modules are typically written as Ansible roles. These test roles are located in `./test/integration/targets`. You must first set up your testing environment, and configure a test inventory for Ansible to connect to.

In this example we will set up a test inventory to connect to two hosts and run the integration tests for win_stat:

- Run the command `source ./hacking/env-setup` to prepare environment.
- Create a copy of `./test/integration/inventory.winrm.template` and name it `inventory.winrm`.
- Fill in entries under `[windows]` and set the required variables that are needed to connect to the host.
- *Install the required Python modules* to support WinRM and a configured authentication method.
- To execute the integration tests, run `ansible-test windows-integration win_stat`; you can replace `win_stat` with the role you want to test.

This will execute all the tests currently defined for that role. You can set the verbosity level using the `-v` argument just as you would with ansible-playbook.

When developing tests for a new module, it is recommended to test a scenario once in check mode and twice not in check mode. This ensures that check mode does not make any changes but reports a change, as well as that the second run is idempotent and does not report changes. For example:

```
- name: remove a file (check mode)
  win_file:
    path: C:\temp
    state: absent
  register: remove_file_check
  check_mode: true

- name: get result of remove a file (check mode)
  win_command: powershell.exe "if (Test-Path -Path 'C:\temp') { 'true' } else { 'false' }
↪"
```

(continues on next page)

(continued from previous page)

```
  register: remove_file_actual_check

- name: assert remove a file (check mode)
  assert:
    that:
    - remove_file_check is changed
    - remove_file_actual_check.stdout == 'true\r\n'

- name: remove a file
  win_file:
    path: C:\temp
    state: absent
  register: remove_file

- name: get result of remove a file
  win_command: powershell.exe "if (Test-Path -Path 'C:\temp') { 'true' } else { 'false' }
↪"
  register: remove_file_actual

- name: assert remove a file
  assert:
    that:
    - remove_file is changed
    - remove_file_actual.stdout == 'false\r\n'

- name: remove a file (idempotent)
  win_file:
    path: C:\temp
    state: absent
  register: remove_file_again

- name: assert remove a file (idempotent)
  assert:
    that:
    - not remove_file_again is changed
```

Windows communication and development support

Join the #ansible-devel or #ansible-windows chat channels (using Matrix at ansible.im or using IRC at irc.libera.chat) for discussions about Ansible development for Windows.

For questions and discussions pertaining to using the Ansible product, use the #ansible channel.

1.17.11 Creating a new collection

Starting with Ansible 2.10, related modules should be developed in a collection. The Ansible core team and community compiled these module development tips and tricks to help companies developing Ansible modules for their products and users developing Ansible modules for third-party products. See *Developing collections* for a more detailed description of the collections format and additional development guidelines.

Note: LICENSING REQUIREMENTS Ansible enforces the following licensing requirements:

- **Utilities (files in `lib/ansible/module_utils/`) may have one of two licenses:**
  - A file in `module_utils` used **only** for a specific vendor's hardware, provider, or service may be licensed under GPLv3+. Adding a new file under `module_utils` with GPLv3+ needs to be approved by the core team.
  - All other `module_utils` must be licensed under BSD, so GPL-licensed third-party and Galaxy modules can use them.
  - If there's doubt about the appropriate license for a file in `module_utils`, the Ansible Core Team will decide during an Ansible Core Community Meeting.
- All other files shipped with Ansible, including all modules, must be licensed under the GPL license (GPLv3 or later).
- Existing license requirements still apply to content in ansible/ansible (ansible-core).
- Content that was previously in ansible/ansible or a collection and has moved to a new collection must retain the license it had in its prior repository.
- Copyright entries by previous committers must also be kept in any moved files.

Before you start coding

This list of prerequisites is designed to help ensure that you develop high-quality modules that work well with ansible-core and provide a seamless user experience.

- Read though all the pages linked off *Developing modules*; paying particular focus to the *Contributing your module to an existing Ansible collection*.
- We encourage PEP 8 compliance. See testing_pep8 for more information.
- We encourage supporting *Python 2.6+ and Python 3.5+*.
- Look at Ansible Galaxy and review the naming conventions in your functional area (such as cloud, networking, databases).

- With great power comes great responsibility: Ansible collection maintainers have a duty to help keep content up to date and release collections they are responsible for regularly. As with all successful community projects, collection maintainers should keep a watchful eye for reported issues and contributions.
- We strongly recommend unit and/or integration tests. Unit tests are especially valuable when external resources (such as cloud or network devices) are required. For more information see *Testing Ansible* and the Testing Working Group.

Naming conventions

Fully Qualified Collection Names (FQCNs) for plugins and modules include three elements:

- the Galaxy namespace, which generally represents the company or group
- the collection name, which generally represents the product or OS
- **the plugin or module name**
  - always in lower case
  - words separated with an underscore (_) character
  - singular, rather than plural, for example, `command` not `commands`

For example, `community.mongodb.mongodb_linux` or `cisco.meraki.meraki_device`.

It is convenient if the organization and repository names on GitHub (or elsewhere) match your namespace and collection names on Ansible Galaxy, but it is not required. The plugin names you select, however, are always the same in your code repository and in your collection artifact on Galaxy.

Speak to us

Circulating your ideas before coding helps you adopt good practices and avoid common mistakes. After reading the "Before you start coding" section you should have a reasonable idea of the structure of your modules. Write a list of your proposed plugin and/or module names, with a short description of what each one does. Circulate that list on IRC or a mailing list so the Ansible community can review your ideas for consistency and familiarity. Names and functionality that are consistent, predictable, and familiar make your collection easier to use.

Where to get support

Ansible has a thriving and knowledgeable community of module developers that is a great resource for getting your questions answered.

In the *Ansible Community Guide* you can find how to:

- Subscribe to the Mailing Lists - We suggest "Ansible Development List" and "Ansible Announce list"
- `#ansible-devel` - We have found that communicating on the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat) works best for developers so we can have an interactive dialogue.
- Working group and other chat channel meetings - Join the various weekly meetings meeting schedule and agenda page

Required files

Your collection should include the following files to be usable:

- an `__init__.py` file - An empty file to initialize namespace and allow Python to import the files. *Required*
- at least one plugin, for example, `/plugins/modules/$your_first_module.py`. *Required*
- if needed, one or more `/plugins/doc_fragments/$topic.py` files - Code documentation, such as details regarding common arguments. *Optional*
- if needed, one or more `/plugins/module_utils/$topic.py` files - Code shared between more than one module, such as common arguments. *Optional*

When you have these files ready, review the *Contributing your module to an existing Ansible collection* again. If you are creating a new collection, you are responsible for all procedures related to your repository, including setting rules for contributions, finding reviewers, and testing and maintaining the code in your collection.

If you need help or advice, consider joining the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat). For more information, see *Where to get support* and *Communicating with the Ansible community*.

New to git or GitHub

We realize this may be your first use of Git or GitHub. The following guides may be of use:

- How to create a fork of ansible/ansible
- How to sync (update) your fork
- How to create a Pull Request (PR)

1.17.12 Testing Ansible

Why test your Ansible contributions?

If you're a developer, one of the most valuable things you can do is to look at GitHub issues and help fix bugs, since bug-fixing is almost always prioritized over feature development. Even for non-developers, helping to test pull requests for bug fixes and features is still immensely valuable.

Ansible users who understand how to write playbooks and roles should be able to test their work. GitHub pull requests will automatically run a variety of tests (for example, Azure Pipelines) that show bugs in action. However, contributors must also test their work outside of the automated GitHub checks and show evidence of these tests in the PR to ensure that their work will be more likely to be reviewed and merged.

Read on to learn how Ansible is tested, how to test your contributions locally, and how to extend testing capabilities.

If you want to learn about testing collections, read *Testing collections*

Types of tests

At a high level we have the following classifications of tests:

sanity

- testing_sanity
- Sanity tests are made up of scripts and tools used to perform static code analysis.
- The primary purpose of these tests is to enforce Ansible coding standards and requirements.

integration

- testing_integration
- Functional tests of modules and Ansible core functionality.

units

- testing_units
- Tests directly against individual parts of the code base.

Testing within GitHub & Azure Pipelines

Organization

When Pull Requests (PRs) are created they are tested using Azure Pipelines, a Continuous Integration (CI) tool. Results are shown at the end of every PR.

When Azure Pipelines detects an error and it can be linked back to a file that has been modified in the PR then the relevant lines will be added as a GitHub comment. For example:

```
The test `ansible-test sanity --test pep8` failed with the following errors:

lib/ansible/modules/network/foo/bar.py:509:17: E265 block comment should start with '# '

The test `ansible-test sanity --test validate-modules` failed with the following error:
lib/ansible/modules/network/foo/bar.py:0:0: E307 version_added should be 2.4. Currently␣
↪2.3
```

From the above example we can see that `--test pep8` and `--test validate-modules` have identified an issue. The commands given allow you to run the same tests locally to ensure you've fixed all issues without having to push your changes to GitHub and wait for Azure Pipelines, for example:

If you haven't already got Ansible available, use the local checkout by running:

```
source hacking/env-setup
```

Then run the tests detailed in the GitHub comment:

```
ansible-test sanity --test pep8
ansible-test sanity --test validate-modules
```

If there isn't a GitHub comment stating what's failed you can inspect the results by clicking on the "Details" button under the "checks have failed" message at the end of the PR.

Rerunning a failing CI job

Occasionally you may find your PR fails due to a reason unrelated to your change. This could happen for several reasons, including:

- a temporary issue accessing an external resource, such as a yum or git repo
- a timeout creating a virtual machine to run the tests on

If either of these issues appear to be the case, you can rerun the Azure Pipelines test by:

- adding a comment with `/rebuild` (full rebuild) or `/rebuild_failed` (rebuild only failed CI nodes) to the PR
- closing and re-opening the PR (full rebuild)
- making another change to the PR and pushing to GitHub

If the issue persists, please contact us in the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat).

How to test a PR

Ideally, code should add tests that prove that the code works. That's not always possible and tests are not always comprehensive, especially when a user doesn't have access to a wide variety of platforms, or is using an API or web service. In these cases, live testing against real equipment can be more valuable than automation that runs against simulated interfaces. In any case, things should always be tested manually the first time as well.

Thankfully, helping to test Ansible is pretty straightforward, assuming you are familiar with how Ansible works.

Setup: Checking out a Pull Request

You can do this by:

- checking out Ansible
- fetching the proposed changes into a test branch
- testing
- commenting on that particular issue on GitHub

Here's how:

Warning: Testing source code from GitHub pull requests sent to us does have some inherent risk, as the source code sent may have mistakes or malicious code that could have a negative impact on your system. We recommend doing all testing on a virtual machine, whether a cloud instance, or locally. Some users like Vagrant or Docker for this, but they are optional. It is also useful to have virtual machines of different Linux or other flavors, since some features (for example, package managers such as apt or yum) are specific to those OS versions.

Create a fresh area to work:

```
git clone https://github.com/ansible/ansible.git ansible-pr-testing
cd ansible-pr-testing
```

Next, find the pull request you'd like to test and make note of its number. It will look something like this:

```
Use os.path.sep instead of hardcoding / #65381
```

Note: Only test `ansible:devel`

It is important that the PR request target be `ansible:devel`, as we do not accept pull requests into any other branch. Dot releases are cherry-picked manually by Ansible staff.

Use the pull request number when you fetch the proposed changes and create your branch for testing:

```
git fetch origin refs/pull/XXXX/head:testing_PRXXXX
git checkout testing_PRXXXX
```

The first command fetches the proposed changes from the pull request and creates a new branch named `testing_PRXXXX`, where the XXXX is the actual number associated with the pull request (for example, 65381). The second command checks out the newly created branch.

Note: If the GitHub user interface shows that the pull request will not merge cleanly, we do not recommend proceeding if you are not somewhat familiar with git and coding, as you will have to resolve a merge conflict. This is the responsibility of the original pull request contributor.

Note: Some users do not create feature branches, which can cause problems when they have multiple, unrelated commits in their version of `devel`. If the source looks like `someuser:devel`, make sure there is only one commit listed on the pull request.

The Ansible source includes a script that allows you to use Ansible directly from source without requiring a full installation that is frequently used by developers on Ansible.

Simply source it (to use the Linux/Unix terminology) to begin using it immediately:

```
source ./hacking/env-setup
```

This script modifies the `PYTHONPATH` environment variables (along with a few other things), which will be temporarily set as long as your shell session is open.

Testing the Pull Request

At this point, you should be ready to begin testing!

Some ideas of what to test are:

- Create a test Playbook with the examples in and check if they function correctly
- Test to see if any Python backtraces returned (that's a bug)
- Test on different operating systems, or against different library versions

Run sanity tests

```
ansible-test sanity
```

More information: testing_sanity

Run unit tests

```
ansible-test units
```

More information: testing_units

Run integration tests

```
ansible-test integration -v ping
```

More information: testing_integration

Any potential issues should be added as comments on the pull request (and it's acceptable to comment if the feature works as well), remembering to include the output of `ansible --version`

Example:

```
Works for me! Tested on `Ansible 2.3.0`.  I verified this on CentOS 6.5 and also Ubuntu␣
↪14.04.
```

If the PR does not resolve the issue, or if you see any failures from the unit/integration tests, just include that output instead:

> This change causes errors for me.
>
> When I ran this Ubuntu 16.04 it failed with the following:
>
> > ```
> > some output
> > StackTrace
> > some other output
> > ```

Code Coverage Online

The online code coverage reports are a good way to identify areas for testing improvement in Ansible. By following red colors you can drill down through the reports to find files which have no tests at all. Adding both integration and unit tests which show clearly how code should work, verify important Ansible functions and increase testing coverage in areas where there is none is a valuable way to help improve Ansible.

The code coverage reports only cover the `devel` branch of Ansible where new feature development takes place. Pull requests and new code will be missing from the codecov.io coverage reports so local reporting is needed. Most `ansible-test` commands allow you to collect code coverage, this is particularly useful to indicate where to extend testing. See testing_running_locally for more information.

Want to know more about testing?

If you'd like to know more about the plans for improving testing Ansible then why not join the Testing Working Group.

1.17.13 The lifecycle of an Ansible module or plugin

Modules and plugins in the main Ansible repository have a defined life cycle, from the first introduction to final removal. The module and plugin lifecycle is tied to the *Ansible release cycle <release_cycle>*. A module or plugin may move through these four stages:

1. When a module or plugin is first accepted into Ansible, we consider it in tech preview and will mark it as such in the documentation.
2. If a module or plugin matures, the 'preview' mark in the documentation is removed. Backward compatibility for these modules and plugins is maintained but not guaranteed, which means their parameters should be maintained with stable meanings.
3. If a module's or plugin's target API changes radically, or if someone creates a better implementation of its functionality, we may mark it deprecated. Modules and plugins that are deprecated are still available but they are reaching the end of their life cycle. We retain deprecated modules and plugins for 4 release cycles with deprecation warnings to help users update playbooks and roles that use them.
4. When a module or plugin has been deprecated for four release cycles, it is removed and replaced with a tombstone entry in the routing configuration. Modules and plugins that are removed are no longer shipped with Ansible. The tombstone entry helps users find alternative modules and plugins.

For modules and plugins in collections, the lifecycle is similar. Since ansible-base 2.10, it is no longer possible to mark modules as 'preview' or 'stable'.

Deprecating modules and plugins in the Ansible main repository

To deprecate a module in ansible-core, you must:

1. Rename the file so it starts with an `_`, for example, rename `old_cloud.py` to `_old_cloud.py`. This keeps the module available and marks it as deprecated on the module index pages.
2. Mention the deprecation in the relevant changelog (by creating a changelog fragment with a section `deprecated_features`).
3. Reference the deprecation in the relevant `porting_guide_core_x.y.rst`.
4. Add `deprecated:` to the documentation with the following sub-values:

removed_in
A `string`, such as `"2.10"`; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4. Mutually exclusive with :removed_by_date:.

remove_by_date
(Added in ansible-base 2.10). An ISO 8601 formatted date when the module will be removed. Usually 2 years from the date the module is deprecated. Mutually exclusive with :removed_in:.

why
Optional string that used to detail why this has been removed.

alternative
Inform users they should do instead, for example, `Use M(whatmoduletouseinstead) instead.`.

- For an example of documenting deprecation, see this PR that deprecates multiple modules. Some of the elements in the PR might now be out of date.

Deprecating modules and plugins in a collection

To deprecate a module in a collection, you must:

1. Add a `deprecation` entry to `plugin_routing` in `meta/runtime.yml`. For example, to deprecate the module `old_cloud`, add:

```
plugin_routing:
    modules:
        old_cloud:
            deprecation:
                removal_version: 2.0.0
                warning_text: Use foo.bar.new_cloud instead.
```

   For other plugin types, you have to replace `modules:` with `<plugin_type>:`, for example `lookup:` for lookup plugins.

   Instead of `removal_version`, you can also use `removal_date` with an ISO 8601 formatted date after which the module will be removed in a new major version of the collection.

2. Mention the deprecation in the relevant changelog. If the collection uses `antsibull-changelog`, create a changelog fragment with a section `deprecated_features`.

3. Add `deprecated:` to the documentation of the module or plugin with the following sub-values:

   removed_in
   A `string`, such as `"2.10"`; the version of Ansible where the module will be replaced with a docs-only module stub. Usually current release +4. Mutually exclusive with :removed_by_date:.

   remove_by_date
   (Added in ansible-base 2.10). An ISO 8601 formatted date when the module will be removed. Usually 2 years from the date the module is deprecated. Mutually exclusive with :removed_in:.

   why
   Optional string that used to detail why this has been removed.

alternative
Inform users they should do instead, for example, `Use M(whatmoduletouseinstead) instead.`.

Changing a module or plugin name in the Ansible main repository

You can also rename a module and keep a deprecated alias to the old name by using a symlink that starts with _. This example allows the `stat` module to be called with `fileinfo`, making the following examples equivalent:

```
ln -s stat.py _fileinfo.py
ansible -m stat -a "path=/tmp" localhost
ansible -m fileinfo -a "path=/tmp" localhost
```

Renaming a module or plugin in a collection, or redirecting a module or plugin to another collection

To rename a module or plugin in a collection, or to redirect a module or plugin to another collection, you need to add a `redirect` entry to `plugin_routing` in `meta/runtime.yml`. For example, to redirect the module `old_cloud` to `foo.bar.new_cloud`, add:

```
plugin_routing:
    modules:
        old_cloud:
            redirect: foo.bar.new_cloud
```

If you want to deprecate the old name, add a `deprecation:` entry (see above):

```
plugin_routing:
    modules:
        old_cloud:
            redirect: foo.bar.new_cloud
            deprecation:
                removal_version: 2.0.0
                warning_text: Use foo.bar.new_cloud instead.
```

You need to use the Fully Qualified Collection Name (FQCN) of the new module/plugin name, even if it is located in the same collection as the redirect. By using a FQCN from another collection, you redirect the module/plugin to that collection.

If you need to support Ansible 2.9, please note that Ansible 2.9 does not know about `meta/runtime.yml`. With Ansible 2.9 you can still rename plugins and modules inside one collection by using symbolic links. Note that ansible-base 2.10, ansible-core 2.11, and newer will prefer `meta/runtime.yml` entries over symbolic links.

Tombstoning a module or plugin in a collection

To remove a deprecated module or plugin from a collection, you need to tombstone it:

1. Remove the module or plugin file with related files like tests, documentation references, and documentation.
2. Add a tombstone entry in `meta/runtime.yml`. For example, to tombstone the module `old_cloud`, add:

```
plugin_routing:
    modules:
        old_cloud:
```

(continues on next page)

(continued from previous page)

```
        tombstone:
          removal_version: 2.0.0
          warning_text: Use foo.bar.new_cloud instead.
```

Instead of `removal_version`, you can also use `removal_date` with an ISO 8601 formatted date. The date should be the date of the next major release.

1.17.14 Developing plugins

Plugins augment Ansible's core functionality with logic and features that are accessible to all modules. Ansible collections include a number of handy plugins, and you can easily write your own. All plugins must:

- be written in Python
- raise errors
- return strings in unicode
- conform to Ansible's configuration and documentation standards

Once you've reviewed these general guidelines, you can skip to the particular type of plugin you want to develop.

Writing plugins in Python

You must write your plugin in Python so it can be loaded by the PluginLoader and returned as a Python object that any module can use. Since your plugin will execute on the controller, you must write it in a *compatible version of Python*.

Raising errors

You should return errors encountered during plugin execution by raising AnsibleError() or a similar class with a message describing the error. When wrapping other exceptions into error messages, you should always use the to_native Ansible function to ensure proper string compatibility across Python versions:

```
from ansible.module_utils.common.text.converters import to_native

try:
    cause_an_exception()
except Exception as e:
    raise AnsibleError('Something happened, this was original exception: %s' % to_
↪native(e))
```

Since Ansible evaluates variables only when they are needed, filter and test plugins should propagate the exceptions jinja2.exceptions.UndefinedError and AnsibleUndefinedVariable to ensure undefined variables are only fatal when necessary.

Check the different AnsibleError objects and see which one applies best to your situation. Check the section on the specific plugin type you're developing for type-specific error handling details.

String encoding

You must convert any strings returned by your plugin into Python's unicode type. Converting to unicode ensures that these strings can run through Jinja2. To convert strings:

```
from ansible.module_utils.common.text.converters import to_text
result_string = to_text(result_string)
```

Plugin configuration & documentation standards

To define configurable options for your plugin, describe them in the DOCUMENTATION section of the python file. Callback and connection plugins have declared configuration requirements this way since Ansible version 2.4; most plugin types now do the same. This approach ensures that the documentation of your plugin's options will always be correct and up-to-date. To add a configurable option to your plugin, define it in this format:

```
options:
  option_name:
    description: describe this config option
    default: default value for this config option
    env:
      - name: NAME_OF_ENV_VAR
    ini:
      - section: section_of_ansible.cfg_where_this_config_option_is_defined
        key: key_used_in_ansible.cfg
    vars:
```

(continues on next page)

(continued from previous page)

```
      - name: name_of_ansible_var
      - name: name_of_second_var
        version_added: X.x
    required: True/False
    type: boolean/float/integer/list/none/path/pathlist/pathspec/string/tmppath
    version_added: X.x
```

To access the configuration settings in your plugin, use `self.get_option(<option_name>)`. Some plugin types hande this differently:

- Become, callback, connection and shell plugins are guaranteed to have the engine call `set_options()`.
- Lookup plugins always require you to handle it in the `run()` method.
- Inventory plugins are done automatically if you use the `base _read_config_file()` method. If not, you must use `self.get_option(<option_name>)`.
- Cache plugins do it on load.
- Cliconf, httpapi and netconf plugins indirectly piggy back on connection plugins.
- Vars plugin settings are populated when first accessed (using the `self.get_option()` or `self.get_options()` method.

If you need to populate settings explicitly, use a `self.set_options()` call.

Configuration sources follow the precedence rules for values in Ansible. When there are multiple values from the same category, the value defined last takes precedence. For example, in the above configuration block, if both `name_of_ansible_var` and `name_of_second_var` are defined, the value of the `option_name` option will be the value of `name_of_second_var`. Refer to *Controlling how Ansible behaves: precedence rules* for further information.

Plugins that support embedded documentation (see *ansible-doc* for the list) should include well-formed doc strings. If you inherit from a plugin, you must document the options it takes, either through a documentation fragment or as a copy. See *Module format and documentation* for more information on correct documentation. Thorough documentation is a good idea even if you're developing a plugin for local use.

In ansible-core 2.14 we added support for documenting filter and test plugins. You have two options for providing documentation:

- Define a Python file that includes inline documentation for each plugin.
- Define a Python file for multiple plugins and create adjacent documentation files in YAML format.

Developing particular plugin types

Action plugins

Action plugins let you integrate local processing and local data with module functionality.

To create an action plugin, create a new class with the Base(ActionBase) class as the parent:

```python
from ansible.plugins.action import ActionBase

class ActionModule(ActionBase):
    pass
```

From there, execute the module using the `_execute_module` method to call the original module. After successful execution of the module, you can modify the module return data.

```
module_return = self._execute_module(module_name='<NAME_OF_MODULE>',
                                     module_args=module_args,
                                     task_vars=task_vars, tmp=tmp)
```

For example, if you wanted to check the time difference between your Ansible controller and your target machine(s), you could write an action plugin to check the local time and compare it to the return data from Ansible's `setup` module:

```
#!/usr/bin/python
# Make coding more python3-ish, this is required for contributions to Ansible
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

from ansible.plugins.action import ActionBase
from datetime import datetime


class ActionModule(ActionBase):
    def run(self, tmp=None, task_vars=None):
        super(ActionModule, self).run(tmp, task_vars)
        module_args = self._task.args.copy()
        module_return = self._execute_module(module_name='setup',
                                             module_args=module_args,
                                             task_vars=task_vars, tmp=tmp)
        ret = dict()
        remote_date = None
        if not module_return.get('failed'):
            for key, value in module_return['ansible_facts'].items():
                if key == 'ansible_date_time':
                    remote_date = value['iso8601']

        if remote_date:
            remote_date_obj = datetime.strptime(remote_date, '%Y-%m-%dT%H:%M:%SZ')
            time_delta = datetime.utcnow() - remote_date_obj
            ret['delta_seconds'] = time_delta.seconds
            ret['delta_days'] = time_delta.days
            ret['delta_microseconds'] = time_delta.microseconds

        return dict(ansible_facts=dict(ret))
```

This code checks the time on the controller, captures the date and time for the remote machine using the `setup` module, and calculates the difference between the captured time and the local time, returning the time delta in days, seconds and microseconds.

For practical examples of action plugins, see the source code for the action plugins included with Ansible Core

Cache plugins

Cache plugins store gathered facts and data retrieved by inventory plugins.

Import cache plugins using the cache_loader so you can use `self.set_options()` and `self.get_option(<option_name>)`. If you import a cache plugin directly in the code base, you can only access options by the `ansible.constants`, and you break the cache plugin's ability to be used by an inventory plugin.

```
from ansible.plugins.loader import cache_loader
[...]
plugin = cache_loader.get('custom_cache', **cache_kwargs)
```

There are two base classes for cache plugins, `BaseCacheModule` for database-backed caches, and `BaseCacheFileModule` for file-backed caches.

To create a cache plugin, start by creating a new `CacheModule` class with the appropriate base class. If you're creating a plugin using an `__init__` method you should initialize the base class with any provided args and kwargs to be compatible with inventory plugin cache options. The base class calls `self.set_options(direct=kwargs)`. After the base class `__init__` method is called `self.get_option(<option_name>)` should be used to access cache options.

New cache plugins should take the options `_uri`, `_prefix`, and `_timeout` to be consistent with existing cache plugins.

```
from ansible.plugins.cache import BaseCacheModule

class CacheModule(BaseCacheModule):
    def __init__(self, *args, **kwargs):
        super(CacheModule, self).__init__(*args, **kwargs)
        self._connection = self.get_option('_uri')
        self._prefix = self.get_option('_prefix')
        self._timeout = self.get_option('_timeout')
```

If you use the `BaseCacheModule`, you must implement the methods `get`, `contains`, `keys`, `set`, `delete`, `flush`, and `copy`. The `contains` method should return a boolean that indicates if the key exists and has not expired. Unlike file-based caches, the `get` method does not raise a KeyError if the cache has expired.

If you use the `BaseFileCacheModule`, you must implement `_load` and `_dump` methods that will be called from the base class methods `get` and `set`.

If your cache plugin stores JSON, use `AnsibleJSONEncoder` in the `_dump` or `set` method and `AnsibleJSONDecoder` in the `_load` or `get` method.

For example cache plugins, see the source code for the cache plugins included with Ansible Core.

Callback plugins

Callback plugins add new behaviors to Ansible when responding to events. By default, callback plugins control most of the output you see when running the command line programs.

To create a callback plugin, create a new class with the Base(Callbacks) class as the parent:

```
from ansible.plugins.callback import CallbackBase

class CallbackModule(CallbackBase):
    pass
```

From there, override the specific methods from the CallbackBase that you want to provide a callback for. For plugins intended for use with Ansible version 2.0 and later, you should only override methods that start with `v2`. For a complete list of methods that you can override, please see `__init__.py` in the lib/ansible/plugins/callback directory.

The following is a modified example of how Ansible's timer plugin is implemented, but with an extra option so you can see how configuration works in Ansible version 2.4 and later:

```
# Make coding more python3-ish, this is required for contributions to Ansible
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

# not only visible to ansible-doc, it also 'declares' the options the plugin requires and␣
↪how to configure them.
DOCUMENTATION = '''
name: timer
callback_type: aggregate
requirements:
    - enable in configuration
short_description: Adds time to play stats
version_added: "2.0"  # for collections, use the collection version, not the Ansible␣
↪version
description:
    - This callback just adds total play duration to the play stats.
options:
  format_string:
    description: format of the string shown to user at play end
    ini:
      - section: callback_timer
        key: format_string
    env:
      - name: ANSIBLE_CALLBACK_TIMER_FORMAT
    default: "Playbook run took %s days, %s hours, %s minutes, %s seconds"
'''
from datetime import datetime

from ansible.plugins.callback import CallbackBase


class CallbackModule(CallbackBase):
    """
    This callback module tells you how long your plays ran for.
    """
    CALLBACK_VERSION = 2.0
    CALLBACK_TYPE = 'aggregate'
    CALLBACK_NAME = 'namespace.collection_name.timer'

    # only needed if you ship it and don't want to enable by default
    CALLBACK_NEEDS_ENABLED = True

    def __init__(self):

      # make sure the expected objects are present, calling the base's __init__
      super(CallbackModule, self).__init__()
```

(continues on next page)

(continued from previous page)

```
        # start the timer when the plugin is loaded, the first play should start a few␣
↪milliseconds after.
        self.start_time = datetime.now()

    def _days_hours_minutes_seconds(self, runtime):
        ''' internal helper method for this callback '''
        minutes = (runtime.seconds // 60) % 60
        r_seconds = runtime.seconds - (minutes * 60)
        return runtime.days, runtime.seconds // 3600, minutes, r_seconds

    # this is only event we care about for display, when the play shows its summary␣
↪stats; the rest are ignored by the base class
    def v2_playbook_on_stats(self, stats):
        end_time = datetime.now()
        runtime = end_time - self.start_time

        # Shows the usage of a config option declared in the DOCUMENTATION variable.␣
↪Ansible will have set it when it loads the plugin.
        # Also note the use of the display object to print to screen. This is available to␣
↪all callbacks, and you should use this over printing yourself
        self._display.display(self._plugin_options['format_string'] % (self._days_hours_
↪minutes_seconds(runtime)))
```

Note that the `CALLBACK_VERSION` and `CALLBACK_NAME` definitions are required for properly functioning plugins for Ansible version 2.0 and later. `CALLBACK_TYPE` is mostly needed to distinguish 'stdout' plugins from the rest, since you can only load one plugin that writes to stdout.

For example callback plugins, see the source code for the callback plugins included with Ansible Core

New in ansible-core 2.11, callback plugins are notified (by the `v2_playbook_on_task_start`) of meta tasks. By default, only explicit `meta` tasks that users list in their plays are sent to callbacks.

There are also some tasks which are generated internally and implicitly at various points in execution. Callback plugins can opt-in to receiving these implicit tasks as well, by setting `self.wants_implicit_tasks = True`. Any `Task` object received by a callback hook will have an `.implicit` attribute, which can be consulted to determine whether the `Task` originated from within Ansible, or explicitly by the user.

Connection plugins

Connection plugins allow Ansible to connect to the target hosts so it can execute tasks on them. Ansible ships with many connection plugins, but only one can be used per host at a time. The most commonly used connection plugins are the `paramiko` SSH, native ssh (just called `ssh`), and `local` connection types. All of these can be used in playbooks and with `/usr/bin/ansible` to connect to remote machines.

Ansible version 2.1 introduced the `smart` connection plugin. The `smart` connection type allows Ansible to automatically select either the `paramiko` or `openssh` connection plugin based on system capabilities, or the `ssh` connection plugin if OpenSSH supports ControlPersist.

To create a new connection plugin (for example, to support SNMP, Message bus, or other transports), copy the format of one of the existing connection plugins and drop it into `connection` directory on your *local plugin path*.

Connection plugins can support common options (such as the `--timeout` flag) by defining an entry in the documentation for the attribute name (in this case `timeout`). If the common option has a non-null default, the plugin should define the same default since a different default would be ignored.

For example connection plugins, see the source code for the connection plugins included with Ansible Core.

Filter plugins

Filter plugins manipulate data. They are a feature of Jinja2 and are also available in Jinja2 templates used by the `template` module. As with all plugins, they can be easily extended, but instead of having a file for each one you can have several per file. Most of the filter plugins shipped with Ansible reside in a `core.py`.

Filter plugins do not use the standard configuration system described above, but since ansible-core 2.14 can use it as plain documentation.

Since Ansible evaluates variables only when they are needed, filter plugins should propagate the exceptions `jinja2.exceptions.UndefinedError` and `AnsibleUndefinedVariable` to ensure undefined variables are only fatal when necessary.

```
try:
    cause_an_exception(with_undefined_variable)
except jinja2.exceptions.UndefinedError as e:
    raise AnsibleUndefinedVariable("Something happened, this was the original exception:
→%s" % to_native(e))
except Exception as e:
    raise AnsibleFilterError("Something happened, this was the original exception: %s" %␣
→to_native(e))
```

For example filter plugins, see the source code for the filter plugins included with Ansible Core.

Inventory plugins

Inventory plugins parse inventory sources and form an in-memory representation of the inventory. Inventory plugins were added in Ansible version 2.4.

You can see the details for inventory plugins in the *Developing dynamic inventory* page.

Lookup plugins

Lookup plugins pull in data from external data stores. Lookup plugins can be used within playbooks both for looping — playbook language constructs like `with_fileglob` and `with_items` are implemented through lookup plugins — and to return values into a variable or parameter.

Lookup plugins are expected to return lists, even if just a single element.

Ansible includes many *filters* which can be used to manipulate the data returned by a lookup plugin. Sometimes it makes sense to do the filtering inside the lookup plugin, other times it is better to return results that can be filtered in the playbook. Keep in mind how the data will be referenced when determining the appropriate level of filtering to be done inside the lookup plugin.

Here's a simple lookup plugin implementation — this lookup returns the contents of a text file as a variable:

```
# python 3 headers, required if submitting to Ansible
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

DOCUMENTATION = r"""
  name: file
```

(continues on next page)

(continued from previous page)

```
  author: Daniel Hokka Zakrisson (@dhozac) <daniel@hozac.com>
  version_added: "0.9"  # for collections, use the collection version, not the Ansible␣
↪version
  short_description: read file contents
  description:
      - This lookup returns the contents from a file on the Ansible controller's file␣
↪system.
  options:
    _terms:
      description: path(s) of files to read
      required: True
    option1:
      description:
            - Sample option that could modify plugin behaviour.
            - This one can be set directly ``option1='x'`` or in ansible.cfg, but can␣
↪also use vars or environment.
      type: string
      ini:
        - section: file_lookup
          key: option1
  notes:
    - if read in variable context, the file can be interpreted as YAML if the content is␣
↪valid to the parser.
    - this lookup does not understand globbing --- use the fileglob lookup instead.
"""
from ansible.errors import AnsibleError, AnsibleParserError
from ansible.plugins.lookup import LookupBase
from ansible.utils.display import Display

display = Display()

class LookupModule(LookupBase):

    def run(self, terms, variables=None, **kwargs):

      # First of all populate options,
      # this will already take into account env vars and ini config
      self.set_options(var_options=variables, direct=kwargs)

      # lookups in general are expected to both take a list as input and output a list
      # this is done so they work with the looping construct 'with_'.
      ret = []
      for term in terms:
          display.debug("File lookup term: %s" % term)

          # Find the file in the expected search path, using a class method
          # that implements the 'expected' search path for Ansible plugins.
          lookupfile = self.find_file_in_search_path(variables, 'files', term)

          # Don't use print or your own logging, the display class
          # takes care of it in a unified way.
          display.vvvv(u"File lookup using %s as file" % lookupfile)
```

(continues on next page)

(continued from previous page)

```
            try:
                if lookupfile:
                    contents, show_data = self._loader._get_file_contents(lookupfile)
                    ret.append(contents.rstrip())
                else:
                    # Always use ansible error classes to throw 'final' exceptions,
                    # so the Ansible engine will know how to deal with them.
                    # The Parser error indicates invalid options passed
                    raise AnsibleParserError()
            except AnsibleParserError:
                raise AnsibleError("could not locate file in lookup: %s" % term)

            # consume an option: if this did something useful, you can retrieve the option␣
↪value here
            if self.get_option('option1') == 'do something':
              pass

        return ret
```

The following is an example of how this lookup is called:

```
---
- hosts: all
  vars:
     contents: "{{ lookup('namespace.collection_name.file', '/etc/foo.txt') }}"
     contents_with_option: "{{ lookup('namespace.collection_name.file', '/etc/foo.txt',␣
↪option1='donothing') }}"
  tasks:

     - debug:
         msg: the value of foo.txt is {{ contents }} as seen today {{ lookup('pipe',
↪'date +"%Y-%m-%d"') }}
```

For example lookup plugins, see the source code for the lookup plugins included with Ansible Core.

For more usage examples of lookup plugins, see *Using Lookups*.

Test plugins

Test plugins verify data. They are a feature of Jinja2 and are also available in Jinja2 templates used by the `template` module. As with all plugins, they can be easily extended, but instead of having a file for each one you can have several per file. Most of the test plugins shipped with Ansible reside in a `core.py`. These are specially useful in conjunction with some filter plugins like `map` and `select`; they are also available for conditional directives like `when:`.

Test plugins do not use the standard configuration system described above. Since ansible-core 2.14 test plugins can use plain documentation.

Since Ansible evaluates variables only when they are needed, test plugins should propagate the exceptions `jinja2.exceptions.UndefinedError` and `AnsibleUndefinedVariable` to ensure undefined variables are only fatal when necessary.

```
try:
    cause_an_exception(with_undefined_variable)
```

(continues on next page)

(continued from previous page)

```
except jinja2.exceptions.UndefinedError as e:
    raise AnsibleUndefinedVariable("Something happened, this was the original exception:
↪%s" % to_native(e))
except Exception as e:
    raise AnsibleFilterError("Something happened, this was the original exception: %s" %␣
↪to_native(e))
```

For example test plugins, see the source code for the test plugins included with Ansible Core.

Vars plugins

Vars plugins inject additional variable data into Ansible runs that did not come from an inventory source, playbook, or command line. Playbook constructs like 'host_vars' and 'group_vars' work using vars plugins.

Vars plugins were partially implemented in Ansible 2.0 and rewritten to be fully implemented starting with Ansible 2.4. Vars plugins are supported by collections starting with Ansible 2.10.

Older plugins used a `run` method as their main body/work:

```
def run(self, name, vault_password=None):
    pass # your code goes here
```

Ansible 2.0 did not pass passwords to older plugins, so vaults were unavailable. Most of the work now happens in the `get_vars` method which is called from the VariableManager when needed.

```
def get_vars(self, loader, path, entities):
    pass # your code goes here
```

The parameters are:

- loader: Ansible's DataLoader. The DataLoader can read files, auto-load JSON/YAML and decrypt vaulted data, and cache read files.
- path: this is 'directory data' for every inventory source and the current play's playbook directory, so they can search for data in reference to them. `get_vars` will be called at least once per available path.
- entities: these are host or group names that are pertinent to the variables needed. The plugin will get called once for hosts and again for groups.

This `get_vars` method just needs to return a dictionary structure with the variables.

Since Ansible version 2.4, vars plugins only execute as needed when preparing to execute a task. This avoids the costly 'always execute' behavior that occurred during inventory construction in older versions of Ansible. Since Ansible version 2.10, vars plugin execution can be toggled by the user to run when preparing to execute a task or after importing an inventory source.

The user must explicitly enable vars plugins that reside in a collection. See *Enabling vars plugins* for details.

Legacy vars plugins are always loaded and run by default. You can prevent them from automatically running by setting `REQUIRES_ENABLED` to True.

```
class VarsModule(BaseVarsPlugin):
    REQUIRES_ENABLED = True
```

Include the `vars_plugin_staging` documentation fragment to allow users to determine when vars plugins run.

```
DOCUMENTATION = '''
    name: custom_hostvars
    version_added: "2.10"  # for collections, use the collection version, not the␣
↪Ansible version
    short_description: Load custom host vars
    description: Load custom host vars
    options:
      stage:
        ini:
          - key: stage
            section: vars_custom_hostvars
        env:
          - name: ANSIBLE_VARS_PLUGIN_STAGE
    extends_documentation_fragment:
      - vars_plugin_staging
'''
```

At times a value provided by a vars plugin will contain unsafe values. The utility function *wrap_var* provided by *ansible.utils.unsafe_proxy* should be used to ensure that Ansible handles the variable and value correctly. The use cases for unsafe data is covered in *Unsafe or raw strings*.

```
from ansible.plugins.vars import BaseVarsPlugin
from ansible.utils.unsafe_proxy import wrap_var

class VarsPlugin(BaseVarsPlugin):
    def get_vars(self, loader, path, entities):
        return dict(
            something_unsafe=wrap_var("{{ SOMETHING_UNSAFE }}")
        )
```

For example vars plugins, see the source code for the vars plugins included with Ansible Core.

See also:

Collection Index
  Browse existing collections, modules, and plugins

Python API
  Learn about the Python API for task execution

Developing dynamic inventory
  Learn about how to develop dynamic inventory sources

Developing modules
  Learn about how to write Ansible modules

Mailing List
  The development mailing list

Real-time chat
  How to join Ansible chat channels

Adjacent YAML documentation files
  Alternate YAML files as documentation

1.17.15 Developing dynamic inventory

Ansible can pull inventory information from dynamic sources, including cloud sources, by using the supplied *inventory plugins*. For details about how to pull inventory information, see *Working with dynamic inventory*. If the source you want is not currently covered by existing plugins, you can create your own inventory plugin as with any other plugin type.

In previous versions, you had to create a script or program that could output JSON in the correct format when invoked with the proper arguments. You can still use and write inventory scripts, as we ensured backwards compatibility through the script inventory plugin and there is no restriction on the programming language used. If you choose to write a script, however, you will need to implement some features yourself such as caching, configuration management, dynamic variable and group composition, and so on. If you use *inventory plugins* instead, you can use the Ansible codebase and add these common features automatically.

Topics

Inventory sources

Inventory sources are the input strings that inventory plugins work with. An inventory source can be a path to a file or to a script, or it can be raw data that the plugin can interpret.

The table below shows some examples of inventory plugins and the source types that you can pass to them with `-i` on the command line.

Plugin	Source
host list	A comma-separated list of hosts
yaml	Path to a YAML format data file
constructed	Path to a YAML configuration file
ini	Path to an INI formatted data file
virtualbox	Path to a YAML configuration file
script plugin	Path to an executable that outputs JSON

Inventory plugins

Like most plugin types (except modules), inventory plugins must be developed in Python. They execute on the controller and should therefore adhere to the *Control node requirements*.

Most of the documentation in *Developing plugins* also applies here. You should read that document first for a general understanding and then come back to this document for specifics on inventory plugins.

Normally, inventory plugins are executed at the start of a run, and before the playbooks, plays, or roles are loaded. However, you can use the `meta: refresh_inventory` task to clear the current inventory and execute the inventory plugins again, and this task will generate a new inventory.

If you use the persistent cache, inventory plugins can also use the configured cache plugin to store and retrieve data. Caching inventory avoids making repeated and costly external calls.

Developing an inventory plugin

The first thing you want to do is use the base class:

```python
from ansible.plugins.inventory import BaseInventoryPlugin

class InventoryModule(BaseInventoryPlugin):

    NAME = 'myplugin'  # used internally by Ansible, it should match the file name but␣
↪not required
```

If the inventory plugin is in a collection, the NAME should be in the 'namespace.collection_name.myplugin' format. The base class has a couple of methods that each plugin should implement and a few helpers for parsing the inventory source and updating the inventory.

After you have the basic plugin working, you can incorporate other features by adding more base classes:

```python
from ansible.plugins.inventory import BaseInventoryPlugin, Constructable, Cacheable

class InventoryModule(BaseInventoryPlugin, Constructable, Cacheable):

    NAME = 'myplugin'
```

For the bulk of the work in a plugin, we mostly want to deal with 2 methods `verify_file` and `parse`.

verify_file method

Ansible uses this method to quickly determine if the inventory source is usable by the plugin. The determination does not need to be 100% accurate, as there might be an overlap in what plugins can handle and by default Ansible will try the enabled plugins as per their sequence.

```python
def verify_file(self, path):
    ''' return true/false if this is possibly a valid file for this plugin to consume '''
    valid = False
    if super(InventoryModule, self).verify_file(path):
        # base class verifies that file exists and is readable by current user
        if path.endswith(('virtualbox.yaml', 'virtualbox.yml', 'vbox.yaml', 'vbox.yml')):
            valid = True
    return valid
```

In the above example, from the virtualbox inventory plugin, we screen for specific file name patterns to avoid attempting to consume any valid YAML file. You can add any type of condition here, but the most common one is 'extension matching'. If you implement extension matching for YAML configuration files, the path suffix <plugin_name>.<yml|yaml> should be accepted. All valid extensions should be documented in the plugin description.

The following is another example that does not use a 'file' but the inventory source string itself, from the host list plugin:

```
def verify_file(self, path):
    ''' don't call base class as we don't expect a path, but a host list '''
    host_list = path
    valid = False
    b_path = to_bytes(host_list, errors='surrogate_or_strict')
    if not os.path.exists(b_path) and ',' in host_list:
        # the path does NOT exist and there is a comma to indicate this is a 'host list'
        valid = True
    return valid
```

This method is just to expedite the inventory process and avoid unnecessary parsing of sources that are easy to filter out before causing a parse error.

parse method

This method does the bulk of the work in the plugin. It takes the following parameters:

- inventory: inventory object with existing data and the methods to add hosts/groups/variables to inventory
- loader: Ansible's DataLoader. The DataLoader can read files, auto load JSON/YAML and decrypt vaulted data, and cache read files.
- path: string with inventory source (this is usually a path, but is not required)
- cache: indicates whether the plugin should use or avoid caches (cache plugin and/or loader)

The base class does some minimal assignment for reuse in other methods.

```
def parse(self, inventory, loader, path, cache=True):

     self.loader = loader
     self.inventory = inventory
     self.templar = Templar(loader=loader)
```

It is up to the plugin now to parse the provided inventory source and translate it into Ansible inventory. To facilitate this, the example below uses a few helper functions:

```
NAME = 'myplugin'

def parse(self, inventory, loader, path, cache=True):

     # call base method to ensure properties are available for use with other helper␣
↪methods
     super(InventoryModule, self).parse(inventory, loader, path, cache)

     # this method will parse 'common format' inventory sources and
     # update any options declared in DOCUMENTATION as needed
     config = self._read_config_data(path)
```

(continues on next page)

(continued from previous page)

```
    # if NOT using _read_config_data you should call set_options directly,
    # to process any defined configuration for this plugin,
    # if you don't define any options you can skip
    #self.set_options()

    # example consuming options from inventory source
    mysession = apilib.session(user=self.get_option('api_user'),
                               password=self.get_option('api_pass'),
                               server=self.get_option('api_server')
    )


    # make requests to get data to feed into inventory
    mydata = mysession.getitall()

    #parse data and create inventory objects:
    for colo in mydata:
        for server in mydata[colo]['servers']:
            self.inventory.add_host(server['name'])
            self.inventory.set_variable(server['name'], 'ansible_host', server[
↪'external_ip'])
```

The specifics will vary depending on API and structure returned. Remember that if you get an inventory source error or any other issue, you should `raise AnsibleParserError` to let Ansible know that the source was invalid or the process failed.

For examples on how to implement an inventory plugin, see the source code here: lib/ansible/plugins/inventory.

inventory object

The `inventory` object passed to `parse` has helpful methods for populating inventory.

`add_group` adds a group to inventory if it doesn't already exist. It takes the group name as the only positional argument.

`add_child` adds a group or host that exists in inventory to a parent group in inventory. It takes two positional arguments, the name of the parent group and the name of the child group or host.

`add_host` adds a host to inventory if it doesn't already exist, optionally to a specific group. It takes the host name as the first argument and accepts two optional keyword arguments, `group` and `port`. `group` is the name of a group in inventory, and `port` is an integer.

`set_variable` adds a variable to a group or host in inventory. It takes three positional arguments: the name of the group or host, the name of the variable, and the value of the variable.

To create groups and variables using Jinja2 expressions, see the section on implementing `constructed` features below.

To see other inventory object methods, see the source code here: lib/ansible/inventory/data.py.

inventory cache

To cache the inventory, extend the inventory plugin documentation with the inventory_cache documentation fragment and use the Cacheable base class.

```
extends_documentation_fragment:
  - inventory_cache
```

```
class InventoryModule(BaseInventoryPlugin, Constructable, Cacheable):

    NAME = 'myplugin'
```

Next, load the cache plugin specified by the user to read from and update the cache. If your inventory plugin uses YAML-based configuration files and the `_read_config_data` method, the cache plugin is loaded within that method. If your inventory plugin does not use `_read_config_data`, you must load the cache explicitly with `load_cache_plugin`.

```
NAME = 'myplugin'

def parse(self, inventory, loader, path, cache=True):
    super(InventoryModule, self).parse(inventory, loader, path)

    self.load_cache_plugin()
```

Before using the cache plugin, you must retrieve a unique cache key by using the `get_cache_key` method. This task needs to be done by all inventory modules using the cache, so that you don't use/overwrite other parts of the cache.

```
def parse(self, inventory, loader, path, cache=True):
    super(InventoryModule, self).parse(inventory, loader, path)

    self.load_cache_plugin()
    cache_key = self.get_cache_key(path)
```

Now that you've enabled caching, loaded the correct plugin, and retrieved a unique cache key, you can set up the flow of data between the cache and your inventory using the `cache` parameter of the `parse` method. This value comes from the inventory manager and indicates whether the inventory is being refreshed (such as by the `--flush-cache` or the meta task `refresh_inventory`). Although the cache shouldn't be used to populate the inventory when being refreshed, the cache should be updated with the new inventory if the user has enabled caching. You can use `self._cache` like a dictionary. The following pattern allows refreshing the inventory to work in conjunction with caching.

```
def parse(self, inventory, loader, path, cache=True):
    super(InventoryModule, self).parse(inventory, loader, path)

    self.load_cache_plugin()
    cache_key = self.get_cache_key(path)

    # cache may be True or False at this point to indicate if the inventory is being␣
↪refreshed
    # get the user's cache option too to see if we should save the cache if it is changing
    user_cache_setting = self.get_option('cache')

    # read if the user has caching enabled and the cache isn't being refreshed
    attempt_to_read_cache = user_cache_setting and cache
```

(continues on next page)

(continued from previous page)

```
    # update if the user has caching enabled and the cache is being refreshed; update␣
↪this value to True if the cache has expired below
    cache_needs_update = user_cache_setting and not cache

    # attempt to read the cache if inventory isn't being refreshed and the user has␣
↪caching enabled
    if attempt_to_read_cache:
        try:
            results = self._cache[cache_key]
        except KeyError:
            # This occurs if the cache_key is not in the cache or if the cache_key␣
↪expired, so the cache needs to be updated
            cache_needs_update = True
    if not attempt_to_read_cache or cache_needs_update:
        # parse the provided inventory source
        results = self.get_inventory()
    if cache_needs_update:
        self._cache[cache_key] = results

    # submit the parsed data to the inventory object (add_host, set_variable, etc)
    self.populate(results)
```

After the `parse` method is complete, the contents of `self._cache` is used to set the cache plugin if the contents of the cache have changed.

You have three other cache methods available:

- `set_cache_plugin` forces the cache plugin to be set with the contents of `self._cache`, before the `parse` method completes
- `update_cache_if_changed` sets the cache plugin only if `self._cache` has been modified, before the `parse` method completes
- `clear_cache` flushes the cache, ultimately by calling the cache plugin's `flush()` method, whose implementation is dependent upon the particular cache plugin in use. Note that if the user is using the same cache backend for facts and inventory, both will get flushed. To avoid this, the user can specify a distinct cache backend in their inventory plugin configuration.

constructed features

Inventory plugins can create host variables and groups from Jinja2 expressions and variables by using features from the `constructed` inventory plugin. To do this, use the `Constructable` base class and extend the inventory plugin's documentation with the `constructed` documentation fragment.

```
extends_documentation_fragment:
  - constructed
```

```
class InventoryModule(BaseInventoryPlugin, Constructable):

    NAME = 'ns.coll.myplugin'
```

There are three main options in the `constructed` documentation fragment:

compose creates variables using Jinja2 expressions. This is implemented by calling the _set_composite_vars method. keyed_groups creates groups of hosts based on variable values. This is implemented by calling the _add_host_to_keyed_groups method. groups creates groups based on Jinja2 conditionals. This is implemented by calling the _add_host_to_composed_groups method.

Each method should be called for every host added to inventory. Three positional arguments are required: the constructed option, a dictionary of variables, and a host name. Calling the method _set_composite_vars first will allow keyed_groups and groups to use the composed variables.

By default, undefined variables are ignored. This is permitted by default for compose so you can make the variable definitions depend on variables that will be populated later in a play from other sources. For groups, it allows using variables that are not always present without having to use the default filter. To support configuring undefined variables to be an error, pass the constructed option strict to each of the methods as a keyword argument.

keyed_groups and groups use any variables already associated with the host (for example, from an earlier inventory source). _add_host_to_keyed_groups and add_host_to_composed_groups can turn this off by passing the keyword argument fetch_hostvars.

Here is an example using all three methods:

```
def add_host(self, hostname, host_vars):
    self.inventory.add_host(hostname, group='all')

    for var_name, var_value in host_vars.items():
        self.inventory.set_variable(hostname, var_name, var_value)

    strict = self.get_option('strict')

    # Add variables created by the user's Jinja2 expressions to the host
    self._set_composite_vars(self.get_option('compose'), host_vars, hostname,␣
↪strict=True)

    # Create user-defined groups using variables and Jinja2 conditionals
    self._add_host_to_composed_groups(self.get_option('groups'), host_vars, hostname,␣
↪strict=strict)
    self._add_host_to_keyed_groups(self.get_option('keyed_groups'), host_vars, hostname,␣
↪strict=strict)
```

By default, group names created with _add_host_to_composed_groups() and _add_host_to_keyed_groups() are valid Python identifiers. Invalid characters are replaced with an underscore _. A plugin can change the sanitization used for the constructed features by setting self._sanitize_group_name to a new function. The core engine also does sanitization, so if the custom function is less strict it should be used in conjunction with the configuration setting TRANSFORM_INVALID_GROUP_CHARS.

```
from ansible.inventory.group import to_safe_group_name

class InventoryModule(BaseInventoryPlugin, Constructable):

    NAME = 'ns.coll.myplugin'

    @staticmethod
    def custom_sanitizer(name):
        return to_safe_group_name(name, replacer='')

    def parse(self, inventory, loader, path, cache=True):
        super(InventoryModule, self).parse(inventory, loader, path)
```

(continues on next page)

(continued from previous page)

```
        self._sanitize_group_name = custom_sanitizer
```

Common format for inventory sources

To simplify development, most plugins use a standard YAML-based configuration file as the inventory source. The file has only one required field `plugin`, which should contain the name of the plugin that is expected to consume the file. Depending on other common features used, you might need other fields, and you can add custom options in each plugin as required. For example, if you use the integrated caching, `cache_plugin`, `cache_timeout` and other cache-related fields could be present.

The 'auto' plugin

From Ansible 2.5 onwards, we include the auto inventory plugin and enable it by default. If the `plugin` field in your standard configuration file matches the name of your inventory plugin, the `auto` inventory plugin will load your plugin. The 'auto' plugin makes it easier to use your plugin without having to update configurations.

Inventory scripts

Even though we now have inventory plugins, we still support inventory scripts, not only for backwards compatibility but also to allow users to use other programming languages.

Inventory script conventions

Inventory scripts must accept the `--list` and `--host <hostname>` arguments. Although other arguments are allowed, Ansible will not use them. Such arguments might still be useful for executing the scripts directly.

When the script is called with the single argument `--list`, the script must output to stdout a JSON object that contains all the groups to be managed. Each group's value should be either an object containing a list of each host, any child groups, and potential group variables, or simply a list of hosts:

```
{
    "group001": {
        "hosts": ["host001", "host002"],
        "vars": {
            "var1": true
        },
        "children": ["group002"]
    },
    "group002": {
        "hosts": ["host003","host004"],
        "vars": {
            "var2": 500
        },
        "children":[]
    }

}
```

If any of the elements of a group are empty, they may be omitted from the output.

When called with the argument `--host <hostname>` (where <hostname> is a host from above), the script must print a JSON object, either empty or containing variables to make them available to templates and playbooks. For example:

```
{
    "VAR001": "VALUE",
    "VAR002": "VALUE"
}
```

Printing variables is optional. If the script does not print variables, it should print an empty JSON object.

Tuning the external inventory script

New in version 1.3.

The stock inventory script system mentioned above works for all versions of Ansible, but calling `--host` for every host can be rather inefficient, especially if it involves API calls to a remote subsystem.

To avoid this inefficiency, if the inventory script returns a top-level element called "_meta", it is possible to return all the host variables in a single script execution. When this meta element contains a value for "hostvars", the inventory script will not be invoked with `--host` for each host. This behavior results in a significant performance increase for large numbers of hosts.

The data to be added to the top-level JSON object looks like this:

```
{

    # results of inventory script as above go here
    # ...

    "_meta": {
        "hostvars": {
            "host001": {
                "var001" : "value"
            },
            "host002": {
                "var002": "value"
            }
        }
    }
}
```

To satisfy the requirements of using `_meta`, to prevent ansible from calling your inventory with `--host` you must at least populate `_meta` with an empty `hostvars` object. For example:

```
{

    # results of inventory script as above go here
    # ...

    "_meta": {
        "hostvars": {}
    }
}
```

If you intend to replace an existing static inventory file with an inventory script, it must return a JSON object which contains an 'all' group that includes every host in the inventory as a member and every group in the inventory as a child. It should also include an 'ungrouped' group which contains all hosts which are not members of any other group. A skeleton example of this JSON object is:

```
{
    "_meta": {
      "hostvars": {}
    },
    "all": {
      "children": [
        "ungrouped"
      ]
    },
    "ungrouped": {
      "children": [
      ]
    }
}
```

An easy way to see how this should look is using *ansible-inventory*, which also supports `--list` and `--host` parameters like an inventory script would.

See also:

Python API
Python API to Playbooks and Ad Hoc Task Execution

Developing modules
Get started with developing a module

Developing plugins
How to develop plugins

AWX
REST API endpoint and GUI for Ansible, syncs with dynamic inventory

Development Mailing List
Mailing list for development topics

Real-time chat
How to join Ansible chat channels

1.17.16 Developing `ansible-core`

Although `ansible-core` (the code hosted in the ansible/ansible repository on GitHub) includes a few plugins that can be swapped out by the playbook directives or configuration, much of the code there is not modular. The documents here give insight into how the parts of `ansible-core` work together.

Ansible module architecture

If you are working on the `ansible-core` code, writing an Ansible module, or developing an action plugin, you may need to understand how Ansible's program flow executes. If you are just using Ansible Modules in playbooks, you can skip this section.

Types of modules

Ansible supports several different types of modules in its code base. Some of these are for backwards compatibility and others are to enable flexibility.

Action plugins

Action plugins look like modules to anyone writing a playbook. Usage documentation for most action plugins lives inside a module of the same name. Some action plugins do all the work, with the module providing only documentation. Some action plugins execute modules. The `normal` action plugin executes modules that don't have special action plugins. Action plugins always execute on the controller.

Some action plugins do all their work on the controller. For example, the debug action plugin (which prints text for the user to see) and the assert action plugin (which tests whether values in a playbook satisfy certain criteria) execute entirely on the controller.

Most action plugins set up some values on the controller, then invoke an actual module on the managed node that does something with these values. For example, the template action plugin takes values from the user to construct a file in a temporary location on the controller using variables from the playbook environment. It then transfers the temporary file to a temporary file on the remote system. After that, it invokes the copy module which operates on the remote system to move the file into its final location, sets file permissions, and so on.

New-style modules

All of the modules that ship with Ansible fall into this category. While you can write modules in any language, all official modules (shipped with Ansible) use either Python or PowerShell.

New-style modules have the arguments to the module embedded inside of them in some manner. Old-style modules must copy a separate file over to the managed node, which is less efficient as it requires two over-the-wire connections instead of only one.

Python

New-style Python modules use the *Ansiballz framework* framework for constructing modules. These modules use imports from `ansible.module_utils` to pull in boilerplate module code, such as argument parsing, formatting of return values as *JSON*, and various file operations.

Note: In Ansible, up to version 2.0.x, the official Python modules used the *Module Replacer framework* framework. For module authors, *Ansiballz framework* is largely a superset of *Module Replacer framework* functionality, so you usually do not need to understand the differences between them.

PowerShell

New-style PowerShell modules use the *Module Replacer framework* framework for constructing modules. These modules get a library of PowerShell code embedded in them before being sent to the managed node.

JSONARGS modules

These modules are scripts that include the string `<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>` in their body. This string is replaced with the JSON-formatted argument string. These modules typically set a variable to that value like this:

```
json_arguments = """<<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>>"""
```

Which is expanded as:

```
json_arguments = """{"param1": "test's quotes", "param2": "\"To be or not to be\" -␣
↪Hamlet"}"""
```

Note: Ansible outputs a *JSON* string with bare quotes. Double quotes are used to quote string values, double quotes inside of string values are backslash escaped, and single quotes may appear unescaped inside of a string value. To use JSONARGS, your scripting language must have a way to handle this type of string. The example uses Python's triple quoted strings to do this. Other scripting languages may have a similar quote character that won't be confused by any quotes in the JSON or it may allow you to define your own start-of-quote and end-of-quote characters. If the language doesn't give you any of these then you'll need to write a *non-native JSON module* or *Old-style module* instead.

These modules typically parse the contents of `json_arguments` using a JSON library and then use them as native variables throughout the code.

Non-native want JSON modules

If a module has the string `WANT_JSON` in it anywhere, Ansible treats it as a non-native module that accepts a filename as its only command-line parameter. The filename is for a temporary file containing a *JSON* string containing the module's parameters. The module needs to open the file, read and parse the parameters, operate on the data, and print its return data as a JSON encoded dictionary to stdout before exiting.

These types of modules are self-contained entities. As of Ansible 2.1, Ansible only modifies them to change a shebang line if present.

See also:

Examples of Non-native modules written in ruby are in the Ansible for Rubyists repository.

Binary modules

From Ansible 2.2 onwards, modules may also be small binary programs. Ansible doesn't perform any magic to make these portable to different systems so they may be specific to the system on which they were compiled or require other binary runtime dependencies. Despite these drawbacks, you may have to compile a custom module against a specific binary library if that's the only way to get access to certain resources.

Binary modules take their arguments and return data to Ansible in the same way as *want JSON modules*.

See also:

One example of a binary module written in go.

Old-style modules

Old-style modules are similar to *want JSON modules*, except that the file that they take contains `key=value` pairs for their parameters instead of *JSON*. Ansible decides that a module is old-style when it doesn't have any of the markers that would show that it is one of the other types.

How modules are executed

When a user uses **`ansible`** or **`ansible-playbook`**, they specify a task to execute. The task is usually the name of a module along with several parameters to be passed to the module. Ansible takes these values and processes them in various ways before they are finally executed on the remote machine.

Executor/task_executor

The TaskExecutor receives the module name and parameters that were parsed from the *playbook* (or from the command-line in the case of **`/usr/bin/ansible`**). It uses the name to decide whether it's looking at a module or an *Action Plugin*. If it's a module, it loads the *Normal Action Plugin* and passes the name, variables, and other information about the task and play to that Action Plugin for further processing.

The `normal` action plugin

The `normal` action plugin executes the module on the remote host. It is the primary coordinator of much of the work to actually execute the module on the managed machine.

- It loads the appropriate connection plugin for the task, which then transfers or executes as needed to create a connection to that host.
- It adds any internal Ansible properties to the module's parameters (for instance, the ones that pass along `no_log` to the module).
- It works with other plugins (connection, shell, become, other action plugins) to create any temporary files on the remote machine and cleans up afterwards.
- It pushes the module and module parameters to the remote host, although the *module_common* code described in the next section decides which format those will take.

- It handles any special cases regarding modules (for instance, async execution, or complications around Windows modules that must have the same names as Python modules, so that internal calling of modules from other Action Plugins work.)

Much of this functionality comes from the *BaseAction* class, which lives in `plugins/action/__init__.py`. It uses the `Connection` and `Shell` objects to do its work.

Note: When *tasks* are run with the `async:` parameter, Ansible uses the `async` Action Plugin instead of the `normal` Action Plugin to invoke it. That program flow is currently not documented. Read the source for information on how that works.

Executor/module_common.py

Code in `executor/module_common.py` assembles the module to be shipped to the managed node. The module is first read in, then examined to determine its type:

- *PowerShell* and *JSON-args modules* are passed through *Module Replacer*.
- New-style *Python modules* are assembled by *Ansiballz framework*.
- *Non-native-want-JSON*, *Binary modules*, and *Old-Style modules* aren't touched by either of these and pass through unchanged.

After the assembling step, one final modification is made to all modules that have a shebang line. Ansible checks whether the interpreter in the shebang line has a specific path configured via an `ansible_$X_interpreter` inventory variable. If it does, Ansible substitutes that path for the interpreter path given in the module. After this, Ansible returns the complete module data and the module type to the *Normal Action* which continues execution of the module.

Assembler frameworks

Ansible supports two assembler frameworks: Ansiballz and the older Module Replacer.

Module Replacer framework

The Module Replacer framework is the original framework implementing new-style modules, and is still used for PowerShell modules. It is essentially a preprocessor (like the C Preprocessor for those familiar with that programming language). It does straight substitutions of specific substring patterns in the module file. There are two types of substitutions:

- Replacements that only happen in the module file. These are public replacement strings that modules can utilize to get helpful boilerplate or access to arguments.
  - `from ansible.module_utils.MOD_LIB_NAME import *` is replaced with the contents of the `ansible/module_utils/MOD_LIB_NAME.py` These should only be used with *new-style Python modules*.
  - `#<<INCLUDE_ANSIBLE_MODULE_COMMON>>` is equivalent to `from ansible.module_utils.basic import *` and should also only apply to new-style Python modules.
  - `# POWERSHELL_COMMON` substitutes the contents of `ansible/module_utils/powershell.ps1`. It should only be used with *new-style Powershell modules*.
- Replacements that are used by `ansible.module_utils` code. These are internal replacement patterns. They may be used internally, in the above public replacements, but shouldn't be used directly by modules.

  - "<<ANSIBLE_VERSION>>" is substituted with the Ansible version. In *new-style Python modules* under the *Ansiballz framework* framework the proper way is to instead instantiate an *AnsibleModule* and then access the version from :attr:`AnsibleModule.ansible_version`.
  - "<<INCLUDE_ANSIBLE_MODULE_COMPLEX_ARGS>>" is substituted with a string which is the Python `repr` of the *JSON* encoded module parameters. Using `repr` on the JSON string makes it safe to embed in a Python file. In new-style Python modules under the Ansiballz framework this is better accessed by instantiating an *AnsibleModule* and then using `AnsibleModule.params`.
  - <<SELINUX_SPECIAL_FILESYSTEMS>> substitutes a string which is a comma-separated list of file systems which have a file system dependent security context in SELinux. In new-style Python modules, if you really need this you should instantiate an *AnsibleModule* and then use `AnsibleModule._selinux_special_fs`. The variable has also changed from a comma-separated string of file system names to an actual python list of file system names.
  - <<INCLUDE_ANSIBLE_MODULE_JSON_ARGS>> substitutes the module parameters as a JSON string. Care must be taken to properly quote the string as JSON data may contain quotes. This pattern is not substituted in new-style Python modules as they can get the module parameters another way.
  - The string `syslog.LOG_USER` is replaced wherever it occurs with the `syslog_facility` which was named in `ansible.cfg` or any `ansible_syslog_facility` inventory variable that applies to this host. In new-style Python modules this has changed slightly. If you really need to access it, you should instantiate an *AnsibleModule* and then use `AnsibleModule._syslog_facility` to access it. It is no longer the actual syslog facility and is now the name of the syslog facility. See the *documentation on internal arguments* for details.

Ansiballz framework

The Ansiballz framework was adopted in Ansible 2.1 and is used for all new-style Python modules. Unlike the Module Replacer, Ansiballz uses real Python imports of things in `ansible/module_utils` instead of merely preprocessing the module. It does this by constructing a zipfile – which includes the module file, files in `ansible/module_utils` that are imported by the module, and some boilerplate to pass in the module's parameters. The zipfile is then Base64 encoded and wrapped in a small Python script which decodes the Base64 encoding and places the zipfile into a temp directory on the managed node. It then extracts just the Ansible module script from the zip file and places that in the temporary directory as well. Then it sets the PYTHONPATH to find Python modules inside of the zip file and imports the Ansible module as the special name, `__main__`. Importing it as `__main__` causes Python to think that it is executing a script rather than simply importing a module. This lets Ansible run both the wrapper script and the module code in a single copy of Python on the remote machine.

Note:

- Ansible wraps the zipfile in the Python script for two reasons:
  - for compatibility with Python 2.6 which has a less functional version of Python's `-m` command-line switch.
  - so that pipelining will function properly. Pipelining needs to pipe the Python module into the Python interpreter on the remote node. Python understands scripts on stdin but does not understand zip files.
- Prior to Ansible 2.7, the module was executed by a second Python interpreter instead of being executed inside of the same process. This change was made once Python-2.4 support was dropped to speed up module execution.

In Ansiballz, any imports of Python modules from the `ansible.module_utils` package trigger inclusion of that Python file into the zipfile. Instances of `#<<INCLUDE_ANSIBLE_MODULE_COMMON>>` in the module are turned into `from ansible.module_utils.basic import *` and `ansible/module-utils/basic.py` is then included in the zipfile. Files that are included from `module_utils` are themselves scanned for imports of other Python modules from `module_utils` to be included in the zipfile as well.

Passing args

Arguments are passed differently by the two frameworks:

- In *Module Replacer framework*, module arguments are turned into a JSON-ified string and substituted into the combined module file.
- In *Ansiballz framework*, the JSON-ified string is part of the script which wraps the zipfile. Just before the wrapper script imports the Ansible module as `__main__`, it monkey-patches the private, `_ANSIBLE_ARGS` variable in `basic.py` with the variable values. When a `ansible.module_utils.basic.AnsibleModule` is instantiated, it parses this string and places the args into `AnsibleModule.params` where it can be accessed by the module's other code.

Warning: If you are writing modules, remember that the way we pass arguments is an internal implementation detail: it has changed in the past and will change again as soon as changes to the common module_utils code allow Ansible modules to forgo using `ansible.module_utils.basic.AnsibleModule`. Do not rely on the internal global `_ANSIBLE_ARGS` variable.

Very dynamic custom modules which need to parse arguments before they instantiate an `AnsibleModule` may use `_load_params` to retrieve those parameters. Although `_load_params` may change in breaking ways if necessary to support changes in the code, it is likely to be more stable than either the way we pass parameters or the internal global variable.

Note: Prior to Ansible 2.7, the Ansible module was invoked in a second Python interpreter and the arguments were then passed to the script over the script's stdin.

Internal arguments

Both *Module Replacer framework* and *Ansiballz framework* send additional arguments to the Ansible module beyond those which the user specified in the playbook. These additional arguments are internal parameters that help implement global Ansible features. Modules often do not need to know about these explicitly because the features are implemented in *`ansible.module_utils.basic`*. However, certain features need support from modules and some knowledge of the internal arguments is useful.

The internal arguments in this section are global. If you need to add a local internal argument to a custom module, create an action plugin for that specific module. See `_original_basename` in the copy action plugin for an example.

_ansible_no_log

Type: `bool`

Set to `True` whenever an argument in a task or play specifies `no_log`. Any module that calls the `AnsibleModule.log()` function handles this action automatically. If you have a module that implements its own logging then you need to check the value of `_ansible_no_log`. To access `_ansible_no_log` in a module, instantiate the `AnsibleModule` utility and then check the value of `AnsibleModule.no_log`.

Note: `no_log` specified in a module's `argument_spec` is handled by a different mechanism.

_ansible_debug

Type: `bool`

Operates verbose logging and logging of external commands that a module executes. If the module uses the `AnsibleModule.debug()` function rather than the `AnsibleModule.log()` function then the messages are only logged if you set the `_ansible_debug` argument to `True`. To access `_ansible_debug` in a module, instantiate the `AnsibleModule` utility and access `AnsibleModule._debug`. For more details, see *DEFAULT_DEBUG*.

_ansible_diff

Type: `bool`

With this parameter you can configure your module to show a unified diff of changes that will be applied to the templated files. To access `_ansible_diff` in a module, instantiate the `AnsibleModule` utility and access `AnsibleModule._diff`. You can also access this parameter using the `diff` keyword in your playbook, or the relevant environment variable. For more details, see *Playbook Keywords* and the *DIFF_ALWAYS* configuration option.

_ansible_verbosity

Type: `int`

You can use this argument to control the level (0 for none) of verbosity in logging.

_ansible_selinux_special_fs

Type: `list` Elements: `strings`

This argument provides modules with the names of file systems which should have a special SELinux context. They are used by the `AnsibleModule` methods which operate on files (changing attributes, moving, and copying).

Most modules can use the built-in `AnsibleModule` methods to manipulate files. To access in a module that needs to know about these special context file systems, instantiate `AnsibleModule` and examine the list in `AnsibleModule._selinux_special_fs`.

This argument replaces `ansible.module_utils.basic.SELINUX_SPECIAL_FS` from *Module Replacer framework*. In the module replacer framework the argument was formatted as a comma-separated string of file system names. Under the Ansiballz framework it is a list. You can access `_ansible_selinux_special_fs` using the corresponding environment variable. For more details, see the *DEFAULT_SELINUX_SPECIAL_FS* configuration option.

New in version 2.1.

_ansible_syslog_facility

This argument controls which syslog facility the module logs to. Most modules should just use the `AnsibleModule.log()` function which will then make use of this. If a module has to use this on its own, it should instantiate the `AnsibleModule` method and then retrieve the name of the syslog facility from `AnsibleModule._syslog_facility`. The Ansiballz code is less elegant than the *Module Replacer framework* code:

```python
# Old module_replacer way
import syslog
syslog.openlog(NAME, 0, syslog.LOG_USER)

```

(continues on next page)

(continued from previous page)

```
# New Ansiballz way
import syslog
facility_name = module._syslog_facility
facility = getattr(syslog, facility_name, syslog.LOG_USER)
syslog.openlog(NAME, 0, facility)
```

For more details, see the *DEFAULT_SYSLOG_FACILITY* configuration option.

New in version 2.1.

_ansible_version

This argument passes the version of Ansible to the module. To access it, a module should instantiate the `AnsibleModule` method and then retrieve the version from `AnsibleModule.ansible_version`. This replaces `ansible.module_utils.basic.ANSIBLE_VERSION` from *Module Replacer framework*.

New in version 2.1.

_ansible_module_name

Type: `str`

This argument passes the information to modules about their name. For more details see, the configuration option *DEFAULT_MODULE_NAME*.

_ansible_string_conversion_action

This argument provides instructions about what modules should do after the values of the user-specified module parameters are converted to strings. For more details, see the *STRING_CONVERSION_ACTION* configuration option.

_ansible_keep_remote_files

Type: `bool`

This argument provides instructions that modules must be ready if they need to keep the remote files. For more details, see the *DEFAULT_KEEP_REMOTE_FILES* configuration option.

_ansible_socket

This argument provides modules with a socket for persistent connections. The argument is created using the *PERSISTENT_CONTROL_PATH_DIR* configuration option.

_ansible_shell_executable

Type: `bool`

This argument ensures that modules use the designated shell executable. For more details, see the *ansible_shell_executable* remote host environment parameter.

_ansible_tmpdir

Type: `str`

This argument provides instructions to modules that all commands must use the designated temporary directory, if created. The action plugin designs this temporary directory.

Modules can access this parameter by using the public `tmpdir` property. The `tmpdir` property will create a temporary directory if the action plugin did not set the parameter.

The directory name is generated randomly, and the the root of the directory is determined by one of these:

- *DEFAULT_LOCAL_TMP*
- remote_tmp
- system_tmpdirs

As a result, using the `ansible.cfg` configuration file to activate or customize this setting will not guarantee that you control the full value.

_ansible_remote_tmp

The module's `tmpdir` property creates a randomized directory name in this directory if the action plugin did not set `_ansible_tmpdir`. For more details, see the remote_tmp parameter of the shell plugin.

Module return values & Unsafe strings

At the end of a module's execution, it formats the data that it wants to return as a JSON string and prints the string to its stdout. The normal action plugin receives the JSON string, parses it into a Python dictionary, and returns it to the executor.

If Ansible templated every string return value, it would be vulnerable to an attack from users with access to managed nodes. If an unscrupulous user disguised malicious code as Ansible return value strings, and if those strings were then templated on the controller, Ansible could execute arbitrary code. To prevent this scenario, Ansible marks all strings inside returned data as `Unsafe`, emitting any Jinja2 templates in the strings verbatim, not expanded by Jinja2.

Strings returned by invoking a module through `ActionPlugin._execute_module()` are automatically marked as `Unsafe` by the normal action plugin. If another action plugin retrieves information from a module through some other means, it must mark its return data as `Unsafe` on its own.

In case a poorly-coded action plugin fails to mark its results as "Unsafe," Ansible audits the results again when they are returned to the executor, marking all strings as `Unsafe`. The normal action plugin protects itself and any other code that it calls with the result data as a parameter. The check inside the executor protects the output of all other action plugins, ensuring that subsequent tasks run by Ansible will not template anything from those results either.

Special considerations

Pipelining

Ansible can transfer a module to a remote machine in one of two ways:

- it can write out the module to a temporary file on the remote host and then use a second connection to the remote host to execute it with the interpreter that the module needs
- or it can use what's known as pipelining to execute the module by piping it into the remote interpreter's stdin.

Pipelining only works with modules written in Python at this time because Ansible only knows that Python supports this mode of operation. Supporting pipelining means that whatever format the module payload takes before being sent over the wire must be executable by Python through stdin.

Why pass args over stdin?

Passing arguments through stdin was chosen for the following reasons:

- When combined with *ANSIBLE_PIPELINING*, this keeps the module's arguments from temporarily being saved onto disk on the remote machine. This makes it harder (but not impossible) for a malicious user on the remote machine to steal any sensitive information that may be present in the arguments.
- Command line arguments would be insecure as most systems allow unprivileged users to read the full commandline of a process.
- Environment variables are usually more secure than the commandline but some systems limit the total size of the environment. This could lead to truncation of the parameters if we hit that limit.

AnsibleModule

Argument spec

The `argument_spec` provided to `AnsibleModule` defines the supported arguments for a module, as well as their type, defaults and more.

Example `argument_spec`:

```
module = AnsibleModule(argument_spec=dict(
    top_level=dict(
        type='dict',
        options=dict(
            second_level=dict(
                default=True,
                type='bool',
            )
        )
    )
))
```

This section will discuss the behavioral attributes for arguments:

type

`type` allows you to define the type of the value accepted for the argument. The default value for `type` is `str`. Possible values are:

- str
- list
- dict
- bool
- int
- float
- path
- raw
- jsonarg
- json
- bytes
- bits

The `raw` type, performs no type validation or type casting, and maintains the type of the passed value.

elements
`elements` works in combination with `type` when `type='list'`. `elements` can then be defined as `elements='int'` or any other type, indicating that each element of the specified list should be of that type.

default
The `default` option allows sets a default value for the argument for the scenario when the argument is not provided to the module. When not specified, the default value is `None`.

fallback
`fallback` accepts a `tuple` where the first argument is a callable (function) that will be used to perform the lookup, based on the second argument. The second argument is a list of values to be accepted by the callable.

The most common callable used is `env_fallback` which will allow an argument to optionally use an environment variable when the argument is not supplied.

Example:

```
username=dict(fallback=(env_fallback, ['ANSIBLE_NET_USERNAME']))
```

choices
`choices` accepts a list of choices that the argument will accept. The types of `choices` should match the `type`.

required
`required` accepts a boolean, either `True` or `False` that indicates that the argument is required. When not specified, `required` defaults to `False`. This should not be used in combination with `default`.

no_log
`no_log` accepts a boolean, either `True` or `False`, that indicates explicitly whether or not the argument value should be masked in logs and output.

Note: In the absence of `no_log`, if the parameter name appears to indicate that the argument value is a password or passphrase (such as "admin_password"), a warning will be shown and the value will be masked in logs but **not** output. To disable the warning and masking for parameters that do not contain sensitive information, set `no_log` to `False`.

aliases
`aliases` accepts a list of alternative argument names for the argument, such as the case where the argument is `name` but the module accepts `aliases=['pkg']` to allow `pkg` to be interchangeably with `name`

options
`options` implements the ability to create a sub-argument_spec, where the sub options of the top level argument are also validated using the attributes discussed in this section. The example at the top of this section demonstrates use of `options`. `type` or `elements` should be `dict` is this case.

apply_defaults
`apply_defaults` works alongside `options` and allows the `default` of the sub-options to be applied even when the top-level argument is not supplied.

In the example of the `argument_spec` at the top of this section, it would allow `module.params['top_level']['second_level']` to be defined, even if the user does not provide `top_level` when calling the module.

removed_in_version
`removed_in_version` indicates which version of ansible-core or a collection a deprecated argument will be removed in. Mutually exclusive with `removed_at_date`, and must be used with `removed_from_collection`.

Example:

```
option = {
  'type': 'str',
  'removed_in_version': '2.0.0',
  'removed_from_collection': 'testns.testcol',
},
```

removed_at_date
`removed_at_date` indicates that a deprecated argument will be removed in a minor ansible-core release or major collection release after this date. Mutually exclusive with `removed_in_version`, and must be used with `removed_from_collection`.

Example:

```
option = {
  'type': 'str',
  'removed_at_date': '2020-12-31',
  'removed_from_collection': 'testns.testcol',
},
```

removed_from_collection
Specifies which collection (or ansible-core) deprecates this deprecated argument. Specify `ansible.builtin` for ansible-core, or the collection's name (format `foo.bar`). Must be used with `removed_in_version` or `removed_at_date`.

deprecated_aliases
Deprecates aliases of this argument. Must contain a list or tuple of dictionaries having some the following keys:

name
The name of the alias to deprecate. (Required.)

version
The version of ansible-core or the collection this alias will be removed in. Either `version` or `date` must be specified.

date
The a date after which a minor release of ansible-core or a major collection release will no longer contain this alias.. Either `version` or `date` must be specified.

collection_name
Specifies which collection (or ansible-core) deprecates this deprecated alias. Specify `ansible.builtin` for ansible-core, or the collection's name (format `foo.bar`). Must be used with `version` or `date`.

Examples:

```
option = {
  'type': 'str',
  'aliases': ['foo', 'bar'],
  'depecated_aliases': [
    {
      'name': 'foo',
      'version': '2.0.0',
      'collection_name': 'testns.testcol',
    },
    {
      'name': 'foo',
      'date': '2020-12-31',
      'collection_name': 'testns.testcol',
    },
  ],
},
```

mutually_exclusive
If `options` is specified, `mutually_exclusive` refers to the sub-options described in `options` and behaves as in *Dependencies between module options*.

required_together
If `options` is specified, `required_together` refers to the sub-options described in `options` and behaves as in *Dependencies between module options*.

required_one_of
If `options` is specified, `required_one_of` refers to the sub-options described in `options` and behaves as in *Dependencies between module options*.

required_if
If `options` is specified, `required_if` refers to the sub-options described in `options` and behaves as in *Dependencies between module options*.

required_by
If `options` is specified, `required_by` refers to the sub-options described in `options` and behaves as in *Dependencies between module options*.

Dependencies between module options

The following are optional arguments for `AnsibleModule()`:

```
module = AnsibleModule(
  argument_spec,
  mutually_exclusive=[
    ('path', 'content'),
  ],
  required_one_of=[
    ('path', 'content'),
  ],
)
```

mutually_exclusive

Must be a sequence (list or tuple) of sequences of strings. Every sequence of strings is a list of option names which are mutually exclusive. If more than one options of a list are specified together, Ansible will fail the module with an error.

Example:

```
mutually_exclusive=[
  ('path', 'content'),
  ('repository_url', 'repository_filename'),
],
```

In this example, the options `path` and `content` must not specified at the same time. Also the options `repository_url` and `repository_filename` must not be specified at the same time. But specifying `path` and `repository_url` is accepted.

To ensure that precisely one of two (or more) options is specified, combine `mutually_exclusive` with `required_one_of`.

required_together

Must be a sequence (list or tuple) of sequences of strings. Every sequence of strings is a list of option names which are must be specified together. If at least one of these options are specified, the other ones from the same sequence must all be present.

Example:

```
required_together=[
  ('file_path', 'file_hash'),
],
```

In this example, if one of the options `file_path` or `file_hash` is specified, Ansible will fail the module with an error if the other one is not specified.

required_one_of

Must be a sequence (list or tuple) of sequences of strings. Every sequence of strings is a list of option names from which at least one must be specified. If none one of these options are specified, Ansible will fail module execution.

Example:

```
required_one_of=[
  ('path', 'content'),
],
```

In this example, at least one of `path` and `content` must be specified. If none are specified, execution will fail. Specifying both is explicitly allowed; to prevent this, combine `required_one_of` with `mutually_exclusive`.

required_if

Must be a sequence of sequences. Every inner sequence describes one conditional dependency. Every sequence must have three or four values. The first two values are the option's name and the option's value which describes the condition. The further elements of the sequence are only needed if the option of that name has precisely this value.

If you want that all options in a list of option names are specified if the condition is met, use one of the following forms:

```
('option_name', option_value, ('option_a', 'option_b', ...)),
('option_name', option_value, ('option_a', 'option_b', ...), False),
```

If you want that at least one option of a list of option names is specified if the condition is met, use the following form:

```
('option_name', option_value, ('option_a', 'option_b', ...), True),
```

Example:

```
required_if=[
  ('state', 'present', ('path', 'content'), True),
  ('force', True, ('force_reason', 'force_code')),
],
```

In this example, if the user specifies `state=present`, at least one of the options `path` and `content` must be supplied (or both). To make sure that precisely one can be specified, combine `required_if` with `mutually_exclusive`.

On the other hand, if `force` (a boolean parameter) is set to `true`, `yes` and so on, both `force_reason` and `force_code` must be specified.

required_by

Must be a dictionary mapping option names to sequences of option names. If the option name in a dictionary key is specified, the option names it maps to must all also be specified. Note that instead of a sequence of option names, you can also specify one single option name.

Example:

```
required_by={
  'force': 'force_reason',
  'path': ('mode', 'owner', 'group'),
},
```

In the example, if `force` is specified, `force_reason` must also be specified. Also, if `path` is specified, then three three options `mode`, `owner` and `group` also must be specified.

Declaring check mode support

To declare that a module supports check mode, supply `supports_check_mode=True` to the `AnsibleModule()` call:

```
module = AnsibleModule(argument_spec, supports_check_mode=True)
```

The module can determine whether it is called in check mode by checking the boolean value `module.check_mode`. If it evaluates to `True`, the module must take care not to do any modification.

If `supports_check_mode=False` is specified, which is the default value, the module will exit in check mode with `skipped=True` and message `remote module (<insert module name here>) does not support check mode`.

Adding file options

To declare that a module should add support for all common file options, supply `add_file_common_args=True` to the `AnsibleModule()` call:

```
module = AnsibleModule(argument_spec, add_file_common_args=True)
```

You can find a list of all file options here. It is recommended that you make your `DOCUMENTATION` extend the doc fragment `ansible.builtin.files` (see *Documentation fragments*) in this case, to make sure that all these fields are correctly documented.

The helper functions `module.load_file_common_arguments()` and `module.set_fs_attributes_if_different()` can be used to handle these arguments for you:

```
argument_spec = {
  'path': {
    'type': 'str',
    'required': True,
  },
}

module = AnsibleModule(argument_spec, add_file_common_args=True)
changed = False

# TODO do something with module.params['path'], like update it's contents

# Ensure that module.params['path'] satisfies the file options supplied by the user
file_args = module.load_file_common_arguments(module.params)
changed = module.set_fs_attributes_if_different(file_args, changed)

module.exit_json(changed=changed)
```

See also:

Python API
    Learn about the Python API for task execution

Developing plugins
    Learn about developing plugins

Mailing List
    The development mailing list

irc.libera.chat
#ansible-devel IRC chat channel

1.17.17 Python API

Topics

Note: This API is intended for internal Ansible use. Ansible may make changes to this API at any time that could break backward compatibility with older versions of the API. Because of this, external use is not supported by Ansible. If you want to use Python API only for executing playbooks or modules, consider ansible-runner first.

There are several ways to use Ansible from an API perspective. You can use the Ansible Python API to control nodes, you can extend Ansible to respond to various Python events, you can write plugins, and you can plug in inventory data from external data sources. This document gives a basic overview and examples of the Ansible execution and playbook API.

If you would like to use Ansible programmatically from a language other than Python, trigger events asynchronously, or have access control and logging demands, please see the AWX project.

Note: Because Ansible relies on forking processes, this API is not thread safe.

Python API example

This example is a simple demonstration that shows how to minimally run a couple of tasks:

```
#!/usr/bin/env python

from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

import json
import shutil

import ansible.constants as C
from ansible.executor.task_queue_manager import TaskQueueManager
from ansible.module_utils.common.collections import ImmutableDict
from ansible.inventory.manager import InventoryManager
from ansible.parsing.dataloader import DataLoader
from ansible.playbook.play import Play
from ansible.plugins.callback import CallbackBase
from ansible.vars.manager import VariableManager
from ansible import context
```

(continues on next page)

(continued from previous page)

```
# Create a callback plugin so we can capture the output
class ResultsCollectorJSONCallback(CallbackBase):
    """A sample callback plugin used for performing an action as results come in.

    If you want to collect all results into a single object for processing at
    the end of the execution, look into utilizing the ``json`` callback plugin
    or writing your own custom callback plugin.
    """

    def __init__(self, *args, **kwargs):
        super(ResultsCollectorJSONCallback, self).__init__(*args, **kwargs)
        self.host_ok = {}
        self.host_unreachable = {}
        self.host_failed = {}

    def v2_runner_on_unreachable(self, result):
        host = result._host
        self.host_unreachable[host.get_name()] = result

    def v2_runner_on_ok(self, result, *args, **kwargs):
        """Print a json representation of the result.

        Also, store the result in an instance attribute for retrieval later
        """
        host = result._host
        self.host_ok[host.get_name()] = result
        print(json.dumps({host.name: result._result}, indent=4))

    def v2_runner_on_failed(self, result, *args, **kwargs):
        host = result._host
        self.host_failed[host.get_name()] = result


def main():
    host_list = ['localhost', 'www.example.com', 'www.google.com']
    # since the API is constructed for CLI it expects certain options to always be set␣
↪in the context object
    context.CLIARGS = ImmutableDict(connection='smart', module_path=['/to/mymodules', '/
↪usr/share/ansible'], forks=10, become=None,
                                    become_method=None, become_user=None, check=False,␣
↪diff=False, verbosity=0)
    # required for
    # https://github.com/ansible/ansible/blob/devel/lib/ansible/inventory/manager.py#L204
    sources = ','.join(host_list)
    if len(host_list) == 1:
        sources += ','

    # initialize needed objects
    loader = DataLoader()  # Takes care of finding and reading yaml, json and ini files
    passwords = dict(vault_pass='secret')

    # Instantiate our ResultsCollectorJSONCallback for handling results as they come in.␣
```

(continues on next page)

(continued from previous page)

```
↪Ansible expects this to be one of its main display outlets
    results_callback = ResultsCollectorJSONCallback()

    # create inventory, use path to host config file as source or hosts in a comma␣
↪separated string
    inventory = InventoryManager(loader=loader, sources=sources)

    # variable manager takes care of merging all the different sources to give you a␣
↪unified view of variables available in each context
    variable_manager = VariableManager(loader=loader, inventory=inventory)

    # instantiate task queue manager, which takes care of forking and setting up all␣
↪objects to iterate over host list and tasks
    # IMPORTANT: This also adds library dirs paths to the module loader
    # IMPORTANT: and so it must be initialized before calling `Play.load()`.
    tqm = TaskQueueManager(
        inventory=inventory,
        variable_manager=variable_manager,
        loader=loader,
        passwords=passwords,
        stdout_callback=results_callback,  # Use our custom callback instead of the␣
↪``default`` callback plugin, which prints to stdout
    )

    # create data structure that represents our play, including tasks, this is basically␣
↪what our YAML loader does internally.
    play_source = dict(
        name="Ansible Play",
        hosts=host_list,
        gather_facts='no',
        tasks=[
            dict(action=dict(module='shell', args='ls'), register='shell_out'),
            dict(action=dict(module='debug', args=dict(msg='{{shell_out.stdout}}'))),
            dict(action=dict(module='command', args=dict(cmd='/usr/bin/uptime'))),
        ]
    )

    # Create play object, playbook objects use .load instead of init or new methods,
    # this will also automatically create the task objects from the info provided in␣
↪play_source
    play = Play().load(play_source, variable_manager=variable_manager, loader=loader)

    # Actually run it
    try:
        result = tqm.run(play)  # most interesting data for a play is actually sent to␣
↪the callback's methods
    finally:
        # we always need to cleanup child procs and the structures we use to communicate␣
↪with them
        tqm.cleanup()
        if loader:
            loader.cleanup_all_tmp_files()
```

(continues on next page)

(continued from previous page)

```
    # Remove ansible tmpdir
    shutil.rmtree(C.DEFAULT_LOCAL_TMP, True)

    print("UP ***********")
    for host, result in results_callback.host_ok.items():
        print('{0} >>> {1}'.format(host, result._result['stdout']))

    print("FAILED *******")
    for host, result in results_callback.host_failed.items():
        print('{0} >>> {1}'.format(host, result._result['msg']))

    print("DOWN *********")
    for host, result in results_callback.host_unreachable.items():
        print('{0} >>> {1}'.format(host, result._result['msg']))


if __name__ == '__main__':
    main()
```

Note: Ansible emits warnings and errors through the display object, which prints directly to stdout, stderr and the Ansible log.

The source code for the `ansible` command line tools (`lib/ansible/cli/`) is available on GitHub.

See also:

Developing dynamic inventory
Developing dynamic inventory integrations

Developing modules
Getting started on developing a module

Developing plugins
How to develop plugins

Development Mailing List
Mailing list for development topics

Real-time chat
How to join Ansible chat channels

1.17.18 Rebasing a pull request

You may find that your pull request (PR) is out-of-date and needs to be rebased. This can happen for several reasons:

- Files modified in your PR are in conflict with changes which have already been merged.
- Your PR is old enough that significant changes to automated test infrastructure have occurred.

Rebasing the branch used to create your PR will resolve both of these issues.

Configuring your remotes

Before you can rebase your PR, you need to make sure you have the proper remotes configured. These instructions apply to any repository on GitHub, including collections repositories. On other platforms (bitbucket, gitlab), the same principles and commands apply but the syntax may be different. We use the ansible/ansible repository here as an example. In other repositories, the branch names may be different. Assuming you cloned your fork in the usual fashion, the `origin` remote will point to your fork:

```
$ git remote -v
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (fetch)
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (push)
```

However, you also need to add a remote which points to the upstream repository:

```
$ git remote add upstream https://github.com/ansible/ansible.git
```

Which should leave you with the following remotes:

```
$ git remote -v
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (fetch)
origin  git@github.com:YOUR_GITHUB_USERNAME/ansible.git (push)
upstream        https://github.com/ansible/ansible.git (fetch)
upstream        https://github.com/ansible/ansible.git (push)
```

Checking the status of your branch should show your fork is up-to-date with the `origin` remote:

```
$ git status
On branch YOUR_BRANCH
Your branch is up-to-date with 'origin/YOUR_BRANCH'.
nothing to commit, working tree clean
```

Rebasing your branch

Once you have an `upstream` remote configured, you can rebase the branch for your PR:

```
$ git pull --rebase upstream devel
```

This will replay the changes in your branch on top of the changes made in the upstream `devel` branch. If there are merge conflicts, you will be prompted to resolve those before you can continue.

After you rebase, the status of your branch changes:

```
$ git status
On branch YOUR_BRANCH
Your branch and 'origin/YOUR_BRANCH' have diverged,
and have 4 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
nothing to commit, working tree clean
```

Don't worry, this is normal after a rebase. You should ignore the `git status` instructions to use `git pull`. We'll cover what to do next in the following section.

Updating your pull request

Now that you've rebased your branch, you need to push your changes to GitHub to update your PR.

Since rebasing re-writes git history, you will need to use a force push:

```
$ git push --force-with-lease
```

Your PR on GitHub has now been updated. This will automatically trigger testing of your changes. You should check in on the status of your PR after tests have completed to see if further changes are required.

Getting help rebasing

For help with rebasing your PR, or other development related questions, join us on the #ansible-devel chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat).

See also:

The Ansible Development Cycle
Information on roadmaps, opening PRs, Ansibullbot, and more

1.17.19 Using and developing module utilities

Ansible provides a number of module utilities, or snippets of shared code, that provide helper functions you can use when developing your own modules. The `basic.py` module utility provides the main entry point for accessing the Ansible library, and all Python Ansible modules must import something from `ansible.module_utils`. A common option is to import `AnsibleModule`:

```
from ansible.module_utils.basic import AnsibleModule
```

The `ansible.module_utils` namespace is not a plain Python package: it is constructed dynamically for each task invocation, by extracting imports and resolving those matching the namespace against a *search path* derived from the active configuration.

To reduce the maintenance burden in a collection or in local modules, you can extract duplicated code into one or more module utilities and import them into your modules. For example, if you have your own custom modules that import a `my_shared_code` library, you can place that into a `./module_utils/my_shared_code.py` file like this:

```
from ansible.module_utils.my_shared_code import MySharedCodeClient
```

When you run `ansible-playbook`, Ansible will merge any files in your local `module_utils` directories into the `ansible.module_utils` namespace in the order defined by the *Ansible search path*.

Naming and finding module utilities

You can generally tell what a module utility does from its name and/or its location. Generic utilities (shared code used by many different kinds of modules) live in the main ansible/ansible codebase, in the `common` subdirectory or in the root directory of `lib/ansible/module_utils`. Utilities used by a particular set of modules generally live in the same collection as those modules. For example:

- `lib/ansible/module_utils/urls.py` contains shared code for parsing URLs
- `openstack.cloud.plugins.module_utils.openstack.py` contains utilities for modules that work with OpenStack instances

- `ansible.netcommon.plugins.module_utils.network.common.config.py` contains utility functions for use by networking modules

Following this pattern with your own module utilities makes everything easy to find and use.

Standard module utilities

Ansible ships with an extensive library of `module_utils` files. You can find the module utility source code in the `lib/ansible/module_utils` directory under your main Ansible path. We describe the most widely used utilities below. For more details on any specific module utility, please see the source code for module_utils.

Note: LICENSING REQUIREMENTS Ansible enforces the following licensing requirements:

- **Utilities (files in `lib/ansible/module_utils/`) may have one of two licenses:**
  - A file in `module_utils` used **only** for a specific vendor's hardware, provider, or service may be licensed under GPLv3+. Adding a new file under `module_utils` with GPLv3+ needs to be approved by the core team.
  - All other `module_utils` must be licensed under BSD, so GPL-licensed third-party and Galaxy modules can use them.
  - If there's doubt about the appropriate license for a file in `module_utils`, the Ansible Core Team will decide during an Ansible Core Community Meeting.
- All other files shipped with Ansible, including all modules, must be licensed under the GPL license (GPLv3 or later).
- Existing license requirements still apply to content in ansible/ansible (ansible-core).
- Content that was previously in ansible/ansible or a collection and has moved to a new collection must retain the license it had in its prior repository.
- Copyright entries by previous committers must also be kept in any moved files.

- `api.py` - Supports generic API modules
- `basic.py` - General definitions and helper utilities for Ansible modules
- `common/dict_transformations.py` - Helper functions for dictionary transformations
- `common/file.py` - Helper functions for working with files
- `common/text/` - Helper functions for converting and formatting text
- `common/parameters.py` - Helper functions for dealing with module parameters
- `common/sys_info.py` - Functions for getting distribution and platform information
- `common/validation.py` - Helper functions for validating module parameters against a module argument spec
- `facts/` - Directory of utilities for modules that return facts. See PR 23012 for more information
- `json_utils.py` - Utilities for filtering unrelated output around module JSON output, like leading and trailing lines
- `powershell/` - Directory of definitions and helper functions for Windows PowerShell modules
- `pycompat24.py` - Exception workaround for Python 2.4
- `service.py` - Utilities to enable modules to work with Linux services (placeholder, not in use)

- `six/__init__.py` - Bundled copy of the Six Python library to aid in writing code compatible with both Python 2 and Python 3
- `splitter.py` - String splitting and manipulation utilities for working with Jinja2 templates
- `urls.py` - Utilities for working with http and https requests

Several commonly-used utilities migrated to collections in Ansible 2.10, including:

- `ismount.py` migrated to `ansible.posix.plugins.module_utils.mount.py` - Single helper function that fixes os.path.ismount
- `known_hosts.py` migrated to `community.general.plugins.module_utils.known_hosts.py` - utilities for working with known_hosts file

For a list of migrated content with destination collections, see the runtime.yml file.

1.17.20 Developing collections

Collections are a distribution format for Ansible content. You can package and distribute playbooks, roles, modules, and plugins using collections. A typical collection addresses a set of related use cases. For example, the `cisco.ios` collection automates management of Cisco IOS devices.

You can create a collection and publish it to Ansible Galaxy or to a private Automation Hub instance. You can publish certified collections to the Red Hat Automation Hub, part of the Red Hat Ansible Automation Platform.

Creating collections

To create a collection:

1. Create a *collection skeleton* with the `collection init` command.
2. Add modules and other content to the collection.
3. Build the collection into a collection artifact with *ansible-galaxy collection build*.
4. Publish the collection artifact to Galaxy with ansible-galaxy collection publish.

A user can then install your collection on their systems.

Creating a collection skeleton

To start a new collection, run the following command in your collections directory:

```
ansible_collections#> ansible-galaxy collection init my_namespace.my_collection
```

Note: Both the namespace and collection names use the same strict set of requirements. See Galaxy namespaces on the Galaxy docsite for those requirements.

It will create the structure `[my_namespace]/[my_collection]/[collection skeleton]`.

Hint: If Git is used for version control, the corresponding repository should be initialized in the collection directory.

Once the skeleton exists, you can populate the directories with the content you want inside the collection. See ansible-collections GitHub Org to get a better idea of what you can place inside a collection.

Reference: the `ansible-galaxy collection` command

Currently the `ansible-galaxy collection` command implements the following sub commands:

- `init`: Create a basic collection skeleton based on the default template included with Ansible or your own template.
- `build`: Create a collection artifact that can be uploaded to Galaxy or your own repository.
- `publish`: Publish a built collection artifact to Galaxy.
- `install`: Install one or more collections.

To learn more about the `ansible-galaxy` command-line tool, see the *ansible-galaxy* man page.

See also:

Using Ansible collections
Learn how to install and use collections.

Collection structure
Directories and files included in the collection skeleton

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Using shared resources in collections

Although developing Ansible modules contained in collections is similar to developing standalone Ansible modules, you use shared resources like documentation fragments and module utilities differently in collections. You can use documentation fragments within and across collections. You can use optional module utilities to support multiple versions of ansible-core in your collection. Collections can also depend on other collections.

Using documentation fragments in collections

To include documentation fragments in your collection:

1. Create the documentation fragment: `plugins/doc_fragments/fragment_name`.
2. Refer to the documentation fragment with its FQCN.

```
extends_documentation_fragment:
  - kubernetes.core.k8s_name_options
  - kubernetes.core.k8s_auth_options
  - kubernetes.core.k8s_resource_options
  - kubernetes.core.k8s_scale_options
```

Documentation fragments covers the basics for documentation fragments. The kubernetes.core collection includes a complete example.

If you use FQCN, you can use documentation fragments from one collection in another collection.

Leveraging optional module utilities in collections

Optional module utilities let you adopt the latest features from the most recent ansible-core release in your collection-based modules without breaking your collection on older Ansible versions. With optional module utilities, you can use the latest features when running against the latest versions, while still providing fallback behaviors when running against older versions.

This implementation, widely used in Python programming, wraps optional imports in conditionals or defensive *try/except* blocks, and implements fallback behaviors for missing imports. Ansible's module payload builder supports these patterns by treating any module_utils import nested in a block (e.g., *if*, *try*) as optional. If the requested import cannot be found during the payload build, it is simply omitted from the target payload and assumed that the importing code will handle its absence at runtime. Missing top-level imports of module_utils packages (imports that are not wrapped in a block statement of any kind) will fail the module payload build, and will not execute on the target.

For example, the *ansible.module_utils.common.respawn* package is only available in Ansible 2.11 and higher. The following module code would fail during the payload build on Ansible 2.10 or earlier (as the requested Python module does not exist, and is not wrapped in a block to signal to the payload builder that it can be omitted from the module payload):

```
from ansible.module_utils.common.respawn import respawn_module
```

By wrapping the import statement in a `try` block, the payload builder will omit the Python module if it cannot be located, and assume that the Ansible module will handle it at runtime:

```
try:
    from ansible.module_utils.common.respawn import respawn_module
except ImportError:
    respawn_module = None
...
if needs_respawn:
    if respawn_module:
        respawn_module(target)
    else:
        module.fail_json('respawn is not available in Ansible < 2.11, ensure that foopkg␣
↪is installed')
```

The optional import behavior also applies to module_utils imported from collections.

Listing collection dependencies

We recommend that collections work as standalone, independent units, depending only on ansible-core. However, if your collection must depend on features and functionality from another collection, list the other collection or collections under `dependencies` in your collection's `galaxy.yml` file. Use the `meta/runtime.yml` file to set the ansible-core version that your collection depends on. For more information on the `galaxy.yml` file, see *Collection Galaxy metadata structure*.

You can use git repositories for collection dependencies during local development and testing. For example:

```
dependencies: {'git@github.com:organization/repo_name.git': 'devel'}
```

Warning: Do not use git repositories as dependencies for published collections. Dependencies for published collections must be other published collections.

See also:

Using Ansible collections
Learn how to install and use collections.

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Testing collections

Testing your collection ensures that your code works well and integrates well with the rest of the Ansible ecosystem. Your collection should pass the sanity tests for Ansible code. You should also add unit tests to cover the code in your collection and integration tests to cover the interactions between your collection and ansible-core.

Testing tools

The main tool for testing collections is `ansible-test`, Ansible's testing tool described in *Testing Ansible* and provided by both the `ansible` and `ansible-core` packages.

You can run several sanity tests, as well as run unit and integration tests for plugins using `ansible-test`. When you test collections, test against the ansible-core version(s) you are targeting.

You must always execute `ansible-test` from the root directory of a collection. You can run `ansible-test` in Docker containers without installing any special requirements. The Ansible team uses this approach in Azure Pipelines both in the ansible/ansible GitHub repository and in the large community collections such as community.general and community.network. The examples below demonstrate running tests in Docker containers.

Sanity tests

To run all sanity tests:

```
ansible-test sanity --docker default -v
```

See testing_sanity for more information. See the full list of sanity tests for details on the sanity tests and how to fix identified issues.

Adding unit tests

You must place unit tests in the appropriate `tests/unit/plugins/` directory. For example, you would place tests for `plugins/module_utils/foo/bar.py` in `tests/unit/plugins/module_utils/foo/test_bar.py` or `tests/unit/plugins/module_utils/foo/bar/test_bar.py`. For examples, see the unit tests in community.general.

To run all unit tests for all supported Python versions:

```
ansible-test units --docker default -v
```

To run all unit tests only for a specific Python version:

```
ansible-test units --docker default -v --python 3.6
```

To run only a specific unit test:

```
ansible-test units --docker default -v --python 3.6 tests/unit/plugins/module_utils/foo/
↪test_bar.py
```

You can specify Python requirements in the `tests/unit/requirements.txt` file. See testing_units for more information, especially on fixture files.

Adding integration tests

You must place integration tests in the appropriate `tests/integration/targets/` directory. For module integration tests, you can use the module name alone. For example, you would place integration tests for `plugins/modules/foo.py` in a directory called `tests/integration/targets/foo/`. For non-module plugin integration tests, you must add the plugin type to the directory name. For example, you would place integration tests for `plugins/connections/bar.py` in a directory called `tests/integration/targets/connection_bar/`. For lookup plugins, the directory must be called `lookup_foo`, for inventory plugins, `inventory_foo`, and so on.

You can write two different kinds of integration tests:

- Ansible role tests run with `ansible-playbook` and validate various aspects of the module. They can depend on other integration tests (usually named `prepare_bar` or `setup_bar`, which prepare a service or install a requirement named `bar` in order to test module `foo`) to set-up required resources, such as installing required libraries or setting up server services.
- `runme.sh` tests run directly as scripts. They can set up inventory files, and execute `ansible-playbook` or `ansible-inventory` with various settings.

For examples, see the integration tests in community.general. See also testing_integration for more details.

Since integration tests can install requirements, and set-up, start and stop services, we recommended running them in docker containers or otherwise restricted environments whenever possible. By default, `ansible-test` supports Docker images for several operating systems. See the list of supported docker images for all options. Use the `default` image mainly for platform-independent integration tests, such as those for cloud modules. The following examples use the `fedora35` image.

To execute all integration tests for a collection:

```
ansible-test integration --docker fedora35 -v
```

If you want more detailed output, run the command with `-vvv` instead of `-v`. Alternatively, specify `--retry-on-error` to automatically re-run failed tests with higher verbosity levels.

To execute only the integration tests in a specific directory:

```
ansible-test integration --docker fedora35 -v connection_bar
```

You can specify multiple target names. Each target name is the name of a directory in `tests/integration/targets/` .

See also:

Testing Ansible
More resources on testing Ansible

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Distributing collections

A collection is a distribution format for Ansible content. A typical collection contains modules and other plugins that address a set of related use cases. For example, a collection might automate administering a particular database. A collection can also contain roles and playbooks.

To distribute your collection and allow others to use it, you can publish your collection on one or more *distribution server*. Distribution servers include:

Distribution server	Collections accepted
Ansible Galaxy	All collections
Pulp 3 Galaxy	All collections, supports signed collections
Red Hat Automation Hub	Only collections certified by Red Hat, supports signed collections
Privately hosted Automation Hub	Collections authorized by the owners

Distributing collections involves four major steps:

1. Initial configuration of your distribution server or servers
2. Building your collection tarball
3. Preparing to publish your collection

4. Publishing your collection

Initial configuration of your distribution server or servers

Configure a connection to one or more distribution servers so you can publish collections there. You only need to configure each distribution server once. You must repeat the other steps (building your collection tarball, preparing to publish, and publishing your collection) every time you publish a new collection or a new version of an existing collection.

1. Create a namespace on each distribution server you want to use.
2. Get an API token for each distribution server you want to use.
3. Specify the API token for each distribution server you want to use.

Creating a namespace

You must upload your collection into a namespace on each distribution server. If you have a login for Ansible Galaxy, your Ansible Galaxy username is usually also an Ansible Galaxy namespace.

Warning: Namespaces on Ansible Galaxy cannot include hyphens. If you have a login for Ansible Galaxy that includes a hyphen, your Galaxy username is not also a Galaxy namespace. For example, `awesome-user` is a valid username for Ansible Galaxy, but it is not a valid namespace.

You can create additional namespaces on Ansible Galaxy if you choose. For Red Hat Automation Hub and private Automation Hub you must create a namespace before you can upload your collection. To create a namespace:

- To create a namespace on Galaxy, see Galaxy namespaces on the Galaxy docsite for details.
- To create a namespace on Red Hat Automation Hub, see the Ansible Certified Content FAQ.

Specify the namespace in the `galaxy.yml` file for each collection. For more information on the `galaxy.yml` file, see *Collection Galaxy metadata structure*.

Getting your API token

An API token authenticates your connection to each distribution server. You need a separate API token for each distribution server. Use the correct API token to connect to each distribution server securely and protect your content.

To get your API token:

- To get an API token for Galaxy, go to the Galaxy profile preferences page and click *API Key*.
- To get an API token for Automation Hub, go to the token page and click *Load token*.

Specifying your API token and distribution server

Each time you publish a collection, you must specify the API token and the distribution server to create a secure connection. You have two options for specifying the token and distribution server:

- You can configure the token in configuration, as part of a `galaxy_server_list` entry in your `ansible.cfg` file. Using configuration is the most secure option.
- You can pass the token at the command line as an argument to the `ansible-galaxy` command. If you pass the token at the command line, you can specify the server at the command line, by using the default setting, or by setting the server in configuration. Passing the token at the command line is insecure, because typing secrets at the command line may expose them to other users on the system.

Specifying the token and distribution server in configuration

By default, Ansible Galaxy is configured as the only distribution server. You can add other distribution servers and specify your API token or tokens in configuration by editing the `galaxy_server_list` section of your `ansible.cfg` file. This is the most secure way to manage authentication for distribution servers. Specify a URL and token for each server. For example:

```
[galaxy]
server_list = release_galaxy

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=abcdefghijklmnopqrtuvwxyz
```

You cannot use `apt-key` with any servers defined in your *galaxy_server_list*. See *Configuring the ansible-galaxy client* for complete details.

Specifying the token at the command line

You can specify the API token at the command line using the `--token` argument of the *ansible-galaxy* command. There are three ways to specify the distribution server when passing the token at the command line:

- using the `--server` argument of the *ansible-galaxy* command
- relying on the default (https://galaxy.ansible.com)
- setting a server in configuration by creating a *GALAXY_SERVER* setting in your `ansible.cfg` file

For example:

```
ansible-galaxy collection publish path/to/my_namespace-my_collection-1.0.0.tar.gz --
↪token abcdefghijklmnopqrtuvwxyz
```

Warning: Using the `--token` argument is insecure. Passing secrets at the command line may expose them to others on the system.

Building your collection tarball

After configuring one or more distribution servers, build a collection tarball. The collection tarball is the published artifact, the object that you upload and other users download to install your collection. To build a collection tarball:

1. Review the version number in your `galaxy.yml` file. Each time you publish your collection, it must have a new version number. You cannot make changes to existing versions of your collection on a distribution server. If you try to upload the same collection version more than once, the distribution server returns the error `Code: conflict.collection_exists`. Collections follow semantic versioning rules. For more information on versions, see *Understanding collection versioning*. For more information on the `galaxy.yml` file, see *Collection Galaxy metadata structure*.
2. Run `ansible-galaxy collection build` from inside the top-level directory of the collection. For example:

```
collection_dir#> ansible-galaxy collection build
```

This command builds a tarball of the collection in the current directory, which you can upload to your selected distribution server:

```
my_collection/
├── galaxy.yml
├── ...
├── my_namespace-my_collection-1.0.0.tar.gz
└── ...
```

Note:

- To reduce the size of collections, certain files and folders are excluded from the collection tarball by default. See *Ignoring files and folders* if your collection directory contains other files you want to exclude.
- The current Galaxy maximum tarball size is 2 MB.

You can upload your tarball to one or more distribution servers. You can also distribute your collection locally by copying the tarball to install your collection directly on target systems.

Ignoring files and folders

You can exclude files from your collection with either *build_ignore* or *Manifest Directives*. For more information on the galaxy.yml file, see *Collection Galaxy metadata structure*.

Include all, with explicit ignores

By default the build step includes all the files in the collection directory in the tarball except for the following:

- galaxy.yml
- *.pyc
- *.retry
- tests/output
- previously built tarballs in the root directory
- various version control directories such as .git/

To exclude other files and folders from your collection tarball, set a list of file glob-like patterns in the build_ignore key in the collection's galaxy.yml file. These patterns use the following special characters for wildcard matching:

- *: Matches everything
- ?: Matches any single character
- [seq]: Matches any character in sequence
- [!seq]:Matches any character not in sequence

For example, to exclude the sensitive folder within the playbooks folder as well any .tar.gz archives, set the following in your galaxy.yml file:

```
build_ignore:
- playbooks/sensitive
- '*.tar.gz'
```

Note: The build_ignore feature is only supported with ansible-galaxy collection build in Ansible 2.10 or newer.

Manifest Directives

New in version 2.14.

The galaxy.yml file supports manifest directives that are historically used in Python packaging, as described in MANIFEST.in commands.

Note: The use of manifest requires installing the optional distlib Python dependency.

Note: The manifest feature is only supported with ansible-galaxy collection build in ansible-core 2.14 or newer, and is mutually exclusive with build_ignore.

For example, to exclude the `sensitive` folder within the `playbooks` folder as well as any `.tar.gz` archives, set the following in your `galaxy.yml` file:

```yaml
manifest:
  directives:
    - recursive-exclude playbooks/sensitive **
    - global-exclude *.tar.gz
```

By default, the `MANIFEST.in` style directives would exclude all files by default, but there are default directives in place. Those default directives are described below. To see the directives in use during build, pass `-vvv` with the `ansible-galaxy collection build` command.

```
include meta/*.yml
include *.txt *.md *.rst COPYING LICENSE
recursive-include tests **
recursive-include docs **.rst **.yml **.yaml **.json **.j2 **.txt
recursive-include roles **.yml **.yaml **.json **.j2
recursive-include playbooks **.yml **.yaml **.json
recursive-include changelogs **.yml **.yaml
recursive-include plugins */**.py
recursive-include plugins/become **.yml **.yaml
recursive-include plugins/cache **.yml **.yaml
recursive-include plugins/callback **.yml **.yaml
recursive-include plugins/cliconf **.yml **.yaml
recursive-include plugins/connection **.yml **.yaml
recursive-include plugins/filter **.yml **.yaml
recursive-include plugins/httpapi **.yml **.yaml
recursive-include plugins/inventory **.yml **.yaml
recursive-include plugins/lookup **.yml **.yaml
recursive-include plugins/netconf **.yml **.yaml
recursive-include plugins/shell **.yml **.yaml
recursive-include plugins/strategy **.yml **.yaml
recursive-include plugins/test **.yml **.yaml
recursive-include plugins/vars **.yml **.yaml
recursive-include plugins/modules **.ps1 **.yml **.yaml
recursive-include plugins/module_utils **.ps1 **.psm1 **.cs
# manifest.directives from galaxy.yml inserted here
exclude galaxy.yml galaxy.yaml MANIFEST.json FILES.json <namespace>-<name>-*.tar.gz
recursive-exclude tests/output **
global-exclude /.* /__pycache__
```

Note: `<namespace>-<name>-*.tar.gz` is expanded with the actual `namespace` and `name`.

The `manifest.directives` supplied in `galaxy.yml` are inserted after the default includes and before the default excludes.

To enable the use of manifest directives without supplying your own, insert either `manifest: {}` or `manifest: null` in the `galaxy.yml` file and remove any use of `build_ignore`.

If the default manifest directives do not meet your needs, you can set `manifest.omit_default_directives` to a value of `true` in `galaxy.yml`. You then must specify a full compliment of manifest directives in `galaxy.yml`. The defaults documented above are a good starting point.

Below is an example where the default directives are not included.

```yaml
manifest:
  directives:
    - include meta/runtime.yml
    - include README.md LICENSE
    - recursive-include plugins */**.py
    - exclude galaxy.yml MANIFEST.json FILES.json <namespace>-<name>-*.tar.gz
    - recursive-exclude tests/output **
  omit_default_directives: true
```

Signing a collection

You can include a GnuPG signature with your collection on a *Pulp 3 Galaxy* server. See Enabling collection signing for details.

You can manually generate detached signatures for a collection using the `gpg` CLI using the following step. This step assume you have generated a GPG private key, but do not cover this process.

```
ansible-galaxy collection build
tar -Oxzf namespace-name-1.0.0.tar.gz MANIFEST.json | gpg --output namespace-name-1.0.0.
↪asc --detach-sign --armor --local-user email@example.com -
```

Preparing to publish your collection

Each time you publish your collection, you must create a *new version* on the distribution server. After you publish a version of a collection, you cannot delete or modify that version. To avoid unnecessary extra versions, check your collection for bugs, typos, and other issues locally before publishing:

1. Install the collection locally.
2. Review the locally installed collection before publishing a new version.

Installing your collection locally

You have two options for installing your collection locally:

- Install your collection locally from the tarball.
- Install your collection locally from your git repository.

Installing your collection locally from the tarball

To install your collection locally from the tarball, run `ansible-galaxy collection install` and specify the collection tarball. You can optionally specify a location using the `-p` flag. For example:

```
collection_dir#> ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.
↪gz -p ./collections
```

Install the tarball into a directory configured in *COLLECTIONS_PATHS* so Ansible can easily find and load the collection. If you do not specify a path value, `ansible-galaxy collection install` installs the collection in the first path defined in *COLLECTIONS_PATHS*.

Installing your collection locally from a git repository

To install your collection locally from a git repository, specify the repository and the branch you want to install:

```
collection_dir#> ansible-galaxy collection install git+https://github.com/org/repo.git,
↪devel
```

You can install a collection from a git repository instead of from Galaxy or Automation Hub. As a developer, installing from a git repository lets you review your collection before you create the tarball and publish the collection. As a user, installing from a git repository lets you use collections or versions that are not in Galaxy or Automation Hub yet.

The repository must contain a `galaxy.yml` or `MANIFEST.json` file. This file provides metadata such as the version number and namespace of the collection.

Installing a collection from a git repository at the command line

To install a collection from a git repository at the command line, use the URI of the repository instead of a collection name or path to a `tar.gz` file. Use the prefix `git+`, unless you're using SSH authentication with the user `git` (for example, `git@github.com:ansible-collections/ansible.windows.git`). You can specify a branch, commit, or tag using the comma-separated git commit-ish syntax.

For example:

```
# Install a collection in a repository using the latest commit on the branch 'devel'
ansible-galaxy collection install git+https://github.com/organization/repo_name.git,devel

# Install a collection from a private github repository
ansible-galaxy collection install git@github.com:organization/repo_name.git

# Install a collection from a local git repository
ansible-galaxy collection install git+file:///home/user/path/to/repo_name.git
```

Warning: Embedding credentials into a git URI is not secure. Use safe authentication options to prevent your credentials from being exposed in logs or elsewhere.

- Use SSH authentication
- Use netrc authentication
- Use http.extraHeader in your git configuration
- Use url.<base>.pushInsteadOf in your git configuration

Specifying the collection location within the git repository

When you install a collection from a git repository, Ansible uses the collection `galaxy.yml` or `MANIFEST.json` metadata file to build the collection. By default, Ansible searches two paths for collection `galaxy.yml` or `MANIFEST.json` metadata files:

- The top level of the repository.
- Each directory in the repository path (one level deep).

If a `galaxy.yml` or `MANIFEST.json` file exists in the top level of the repository, Ansible uses the collection metadata in that file to install an individual collection.

```
├── galaxy.yml
├── plugins/
│   ├── lookup/
│   ├── modules/
│   └── module_utils/
└─── README.md
```

If a `galaxy.yml` or `MANIFEST.json` file exists in one or more directories in the repository path (one level deep), Ansible installs each directory with a metadata file as a collection. For example, Ansible installs both collection1 and collection2 from this repository structure by default:

```
├── collection1
│   ├── docs/
│   ├── galaxy.yml
│   └── plugins/
│       ├── inventory/
│       └── modules/
└── collection2
    ├── docs/
    ├── galaxy.yml
    ├── plugins/
    |   ├── filter/
    |   └── modules/
    └── roles/
```

If you have a different repository structure or only want to install a subset of collections, you can add a fragment to the end of your URI (before the optional comma-separated version) to indicate the location of the metadata file or files. The path should be a directory, not the metadata file itself. For example, to install only collection2 from the example repository with two collections:

```
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪collection2/
```

In some repositories, the main directory corresponds to the namespace:

```
namespace/
├── collectionA/
|   ├── docs/
|   ├── galaxy.yml
|   ├── plugins/
|   │   ├── README.md
|   │   └── modules/
|   ├── README.md
|   └── roles/
└── collectionB/
    ├── docs/
    ├── galaxy.yml
    ├── plugins/
    │   ├── connection/
    │   └── modules/
    ├── README.md
    └── roles/
```

You can install all collections in this repository, or install one collection from a specific commit:

```
# Install all collections in the namespace
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪namespace/

# Install an individual collection using a specific commit
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪namespace/collectionA/,7b60ddc245bc416b72d8ea6ed7b799885110f5e5
```

Reviewing your collection

Review the collection:

- Run a playbook that uses the modules and plugins in your collection. Verify that new features and functionality work as expected. For examples and more details see *Using collections*.
- Check the documentation for typos.
- Check that the version number of your tarball is higher than the latest published version on the distribution server or servers.
- If you find any issues, fix them and rebuild the collection tarball.

Understanding collection versioning

The only way to change a collection is to release a new version. The latest version of a collection (by highest version number) is the version displayed everywhere in Galaxy and Automation Hub. Users can still download older versions.

Follow semantic versioning when setting the version for your collection. In summary:

- Increment the major version number, `x` of `x.y.z`, for an incompatible API change.
- Increment the minor version number, `y` of `x.y.z`, for new functionality in a backwards compatible manner (for example new modules/plugins, parameters, return values).
- Increment the patch version number, `z` of `x.y.z`, for backwards compatible bug fixes.

Read the official Semantic Versioning documentation for details and examples.

Publishing your collection

The last step in distributing your collection is publishing the tarball to Ansible Galaxy, Red Hat Automation Hub, or a privately hosted Automation Hub instance. You can publish your collection in two ways:

- from the command line using the `ansible-galaxy collection publish` command
- from the website of the distribution server (Galaxy, Automation Hub) itself

Publishing a collection from the command line

To upload the collection tarball from the command line using `ansible-galaxy`:

```
ansible-galaxy collection publish path/to/my_namespace-my_collection-1.0.0.tar.gz
```

Note: This ansible-galaxy command assumes you have retrieved and stored your API token in configuration. See *Specifying your API token and distribution server* for details.

The `ansible-galaxy collection publish` command triggers an import process, just as if you uploaded the collection through the Galaxy website. The command waits until the import process completes before reporting the status back. If you want to continue without waiting for the import result, use the `--no-wait` argument and manually look at the import progress in your My Imports page.

Publishing a collection from the website

To publish your collection directly on the Galaxy website:

1. Go to the My Content page, and click the **Add Content** button on one of your namespaces.
2. From the **Add Content** dialogue, click **Upload New Collection**, and select the collection archive file from your local filesystem.

When you upload a collection, Ansible always uploads the tarball to the namespace specified in the collection metadata in the `galaxy.yml` file, no matter which namespace you select on the website. If you are not an owner of the namespace specified in your collection metadata, the upload request fails.

After Galaxy uploads and accepts a collection, the website shows you the **My Imports** page. This page shows import process information. You can review any errors or warnings about your upload there.

See also:

Using Ansible collections
Learn how to install and use collections.

Collection Galaxy metadata structure
Table of fields used in the `galaxy.yml` file

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Documenting collections

Documenting modules and plugins

Documenting modules is thoroughly documented in *Module format and documentation*. Plugins can be documented the same way as modules, that is with `DOCUMENTATION`, `EXAMPLES`, and `RETURN` blocks.

Documenting roles

To document a role, you have to add a role argument spec by creating a file `meta/argument_specs.yml` in your role. See *Role argument validation* for details. As an example, you can look at the argument specs file of the sensu.sensu_go.install role on GitHub.

Build a docsite with antsibull-docs

You can use antsibull-docs to build a Sphinx-based docsite for your collection:

1. Create your collection and make sure you can use it with ansible-core by adding it to your *COLLECTIONS_PATHS*.
2. Create a directory `dest` and run `antsibull-docs sphinx-init --use-current --dest-dir dest namespace.name`, where `namespace.name` is the name of your collection.
3. Go into `dest` and run `pip install -r requirements.txt`. You might want to create a venv and activate it first to avoid installing this globally.
4. Then run `./build.sh`.
5. Open `build/html/index.html` in a browser of your choice.

If you want to add additional documentation to your collection next to the plugin, module, and role documentation, see *docs directory*.

Migrating Ansible content to a different collection

When you move content from one collection to another, for example to extract a set of related modules out of `community.general` to create a more focused collection, you must make sure the transition is easy for users to follow.

Migrating content

Before you start migrating content from one collection to another, look at Ansible Collection Checklist.

To migrate content from one collection to another, if the collections are parts of Ansible distribution:

1. Copy content from the source (old) collection to the target (new) collection.
2. Deprecate the module/plugin with `removal_version` scheduled for the next major version in `meta/runtime.yml` of the old collection. The deprecation must be released after the copied content has been included in a release of the new collection.
3. When the next major release of the old collection is prepared:

- remove the module/plugin from the old collection
- remove the symlink stored in `plugin/modules` directory if appropriate (mainly when removing from `community.general` and `community.network`)

- remove related unit and integration tests
- remove specific module utils
- remove specific documentation fragments if there are any in the old collection
- add a changelog fragment containing entries for `removed_features` and `breaking_changes`; you can see an example of a changelog fragment in this pull request
- change `meta/runtime.yml` in the old collection:
  - add `redirect` to the corresponding module/plugin's entry
  - in particular, add `redirect` for the removed module utils and documentation fragments if applicable
  - remove `removal_version` from there
- remove related entries from `tests/sanity/ignore.txt` files if exist
- remove changelog fragments for removed content that are not yet part of the changelog (in other words, do not modify *changelogs/changelog.yaml* and do not delete files mentioned in it)
- remove requirements that are no longer required in `tests/unit/requirements.txt`, `tests/requirements.yml` and `galaxy.yml`

To implement these changes, you need to create at least three PRs:

1. Create a PR against the new collection to copy the content.
2. Deprecate the module/plugin in the old collection.
3. Later create a PR against the old collection to remove the content according to the schedule.

Adding the content to the new collection

Create a PR in the new collection to:

1. Copy ALL the related files from the old collection.
2. If it is an action plugin, include the corresponding module with documentation.
3. If it is a module, check if it has a corresponding action plugin that should move with it.
4. Check `meta/` for relevant updates to `runtime.yml` if it exists.
5. Carefully check the moved `tests/integration` and `tests/units` and update for FQCN.
6. Review `tests/sanity/ignore-*.txt` entries in the old collection.
7. Update `meta/runtime.yml` in the old collection.

Removing the content from the old collection

Create a PR against the source collection repository to remove the modules, module_utils, plugins, and docs_fragments related to this migration:

1. If you are removing an action plugin, remove the corresponding module that contains the documentation.
2. If you are removing a module, remove any corresponding action plugin that should stay with it.
3. Remove any entries about removed plugins from `meta/runtime.yml`. Ensure they are added into the new repo.
4. Remove sanity ignore lines from `tests/sanity/ignore\*.txt`

5. Remove associated integration tests from `tests/integrations/targets/` and unit tests from `tests/units/plugins/`.
6. if you are removing from content from `community.general` or `community.network`, remove entries from `.github/BOTMETA.yml`.
7. Carefully review `meta/runtime.yml` for any entries you may need to remove or update, in particular deprecated entries.
8. Update `meta/runtime.yml` to contain redirects for EVERY PLUGIN, pointing to the new collection name.

Warning: Maintainers for the old collection have to make sure that the PR is merged in a way that it does not break user experience and semantic versioning:

1. A new version containing the merged PR must not be released before the collection the content has been moved to has been released again, with that content contained in it. Otherwise the redirects cannot work and users relying on that content will experience breakage.
2. Once 1.0.0 of the collection from which the content has been removed has been released, such PRs can only be merged for a new **major** version (in other words, 2.0.0, 3.0.0, and so on).

Updating BOTMETA.yml

The `BOTMETA.yml`, for example in community.general collection repository, is the source of truth for:

- ansibullbot

If the old and/or new collection has `ansibullbot`, its `BOTMETA.yml` must be updated correspondingly.

Ansibulbot will know how to redirect existing issues and PRs to the new repo. The build process for docs.ansible.com will know where to find the module docs.

```yaml
$modules/monitoring/grafana/grafana_plugin.py:
    migrated_to: community.grafana
$modules/monitoring/grafana/grafana_dashboard.py:
    migrated_to: community.grafana
$modules/monitoring/grafana/grafana_datasource.py:
    migrated_to: community.grafana
$plugins/callback/grafana_annotations.py:
    maintainers: $team_grafana
    labels: monitoring grafana
    migrated_to: community.grafana
$plugins/doc_fragments/grafana.py:
    maintainers: $team_grafana
    labels: monitoring grafana
    migrated_to: community.grafana
```

Example PR

- The `migrated_to:` key must be added explicitly for every *file*. You cannot add `migrated_to` at the directory level. This is to allow module and plugin webdocs to be redirected to the new collection docs.
- `migrated_to:` MUST be added for every:
  - module
  - plugin

  - module_utils
  - contrib/inventory script
- You do NOT need to add `migrated_to` for:
  - Unit tests
  - Integration tests
  - ReStructured Text docs (anything under `docs/docsite/rst/`)
  - Files that never existed in `ansible/ansible:devel`

See also:

Using Ansible collections
Learn how to install and use collections.

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Contributing to collections

If you want to add functionality to an existing collection, modify a collection you are using to fix a bug, or change the behavior of a module in a collection, clone the git repository for that collection and make changes on a branch. You can combine changes to a collection with a local checkout of Ansible (`source hacking/env-setup`). You should first check the collection repository to see if it has specific contribution guidelines. These are typically listed in the README.md or CONTRIBUTING.md files within the repository.

Contributing to a collection: community.general

These instructions apply to collections hosted in the ansible_collections GitHub organization. For other collections, especially for collections not hosted on GitHub, check the `README.md` of the collection for information on contributing to it.

This example uses the community.general collection. To contribute to other collections in the same GitHub org, replace the folder names `community` and `general` with the namespace and collection name of a different collection.

Prerequisites

- Include `~/dev/ansible/collections/` in *COLLECTIONS_PATHS*
- If that path mentions multiple directories, make sure that no other directory earlier in the search path contains a copy of `community.general`.

Creating a PR

- Create the directory `~/dev/ansible/collections/ansible_collections/community`:

```
mkdir -p ~/dev/ansible/collections/ansible_collections/community
```

- Clone the community.general Git repository or a fork of it into the directory `general`:

```
cd ~/dev/ansible/collections/ansible_collections/community
git clone git@github.com:ansible-collections/community.general.git general
```

- If you clone from a fork, add the original repository as a remote `upstream`:

```
cd ~/dev/ansible/collections/ansible_collections/community/general
git remote add upstream git@github.com:ansible-collections/community.general.git
```

- Create a branch and commit your changes on the branch.
- Remember to add tests for your changes, see *Testing collections*.
- Push your changes to your fork of the collection and create a Pull Request.

You can test your changes by using this checkout of `community.general` in playbooks and roles with whichever version of Ansible you have installed locally, including a local checkout of `ansible/ansible`'s `devel` branch.

See also:

Using Ansible collections
Learn how to install and use collections.

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Generating changelogs and porting guide entries in a collection

You can create and share changelog and porting guide entries for your collection. If your collection is part of the Ansible Community package, we recommend that you use the antsibull-changelog tool to generate Ansible-compatible changelogs. The Ansible changelog uses the output of this tool to collate all the collections included in an Ansible release into one combined changelog for the release.

Note: Ansible here refers to the Ansible 2.10 or later release that includes a curated set of collections.

Understanding antsibull-changelog

The `antsibull-changelog` tool allows you to create and update changelogs for Ansible collections that are compatible with the combined Ansible changelogs. This is an update to the changelog generator used in prior Ansible releases. The tool adds three new changelog fragment categories: `breaking_changes`, `security_fixes` and `trivial`. The tool also generates the `changelog.yaml` file that Ansible uses to create the combined `CHANGELOG.rst` file and Porting Guide for the release.

See *Creating a changelog fragment* and the antsibull-changelog documentation for complete details.

Note: The collection maintainers set the changelog policy for their collections. See the individual collection contributing guidelines for complete details.

Generating changelogs

To initialize changelog generation:

1. Install `antsibull-changelog`: `pip install antsibull-changelog`.
2. Initialize changelogs for your repository: `antsibull-changelog init <path/to/your/collection>`.
3. Optionally, edit the `changelogs/config.yaml` file to customize the location of the generated changelog `.rst` file or other options. See Bootstrapping changelogs for collections for details.

To generate changelogs from the changelog fragments you created:

1. Optionally, validate your changelog fragments: `antsibull-changelog lint`.
2. Generate the changelog for your release: `antsibull-changelog release [--version version_number]`.

Note: Add the `--reload-plugins` option if you ran the `antsibull-changelog release` command previously and the version of the collection has not changed. `antsibull-changelog` caches the information on all plugins and does not update its cache until the collection version changes.

Porting Guide entries from changelog fragments

The Ansible changelog generator automatically adds several changelog fragment categories to the Ansible Porting Guide:

- `major_changes`
- `breaking_changes`
- `deprecated_features`
- `removed_features`

Including collection changelogs into Ansible

If your collection is part of Ansible, use one of the following three options to include your changelog into the Ansible release changelog:

- Use the `antsibull-changelog` tool.
- If are not using this tool, include the properly formatted `changelog.yaml` file into your collection. See the changelog.yaml format for details.
- Add a link to own changelogs or release notes in any format by opening an issue at https://github.com/ansible-community/ansible-build-data/ with the HTML link to that information.

Note: For the first two options, Ansible pulls the changelog details from Galaxy so your changelogs must be included in the collection version on Galaxy that is included in the upcoming Ansible release.

See also:

Generating changelogs and porting guide entries in a collection
Learn how to create good changelog fragments.

Using Ansible collections
Learn how to install and use collections.

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Collection structure

A collection is a simple data structure. None of the directories are required unless you have specific content that belongs in one of them. A collection does require a `galaxy.yml` file at the root level of the collection. This file contains all of the metadata that Galaxy and other tools need in order to package, build and publish the collection.

Collection directories and files

A collection can contain these directories and files:

```
collection/
├── docs/
├── galaxy.yml
├── meta/
│   └── runtime.yml
├── plugins/
│   ├── modules/
│   │   └── module1.py
│   ├── inventory/
│   └── .../
├── README.md
├── roles/
│   ├── role1/
│   ├── role2/
│   └── .../
├── playbooks/
│   ├── files/
│   ├── vars/
│   ├── templates/
│   └── tasks/
└── tests/
```

Note:

- Ansible only accepts `.md` extensions for the `README` file and any files in the `/docs` folder.
- See the ansible-collections GitHub Org for examples of collection structure.
- Not all directories are currently in use. Those are placeholders for future features.

galaxy.yml

A collection must have a `galaxy.yml` file that contains the necessary information to build a collection artifact. See *Collection Galaxy metadata structure* for details.

docs directory

Use the `docs` folder to describe how to use the roles and plugins the collection provides, role requirements, and so on.

For certified collections, Automation Hub displays documents written in markdown in the main `docs` directory with no subdirectories. This will not display on https://docs.ansible.com.

For community collections included in the Ansible PyPI package, docs.ansible.com displays documents written in reStructuredText (.rst) in a docsite/rst/ subdirectory. Define the structure of your extra documentation in `docs/docsite/extra-docs.yml`:

```
---
sections:
```

(continues on next page)

(continued from previous page)

```
- title: Scenario Guide
  toctree:
    - scenario_guide
```

The index page of the documentation for your collection displays the title you define in `docs/docsite/extra-docs.yml` with a link to your extra documentation. For an example, see the community.docker collection repo and the community.docker collection documentation.

You can add extra links to your collection index page and plugin pages with the `docs/docsite/links.yml` file. This populates the links under Description and Communications headings as well as links at the end of the individual plugin pages. See the collection_template links.yml file for a complete description of the structure and use of this file to create links.

Plugin and module documentation

Keep the specific documentation for plugins and modules embedded as Python docstrings. Use `ansible-doc` to view documentation for plugins inside a collection:

```
ansible-doc -t lookup my_namespace.my_collection.lookup1
```

The `ansible-doc` command requires the fully qualified collection name (FQCN) to display specific plugin documentation. In this example, `my_namespace` is the Galaxy namespace and `my_collection` is the collection name within that namespace.

Note: The Galaxy namespace of an Ansible collection is defined in the `galaxy.yml` file. It can be different from the GitHub organization or repository name.

plugins directory

Add a 'per plugin type' specific subdirectory here, including `module_utils` which is usable not only by modules, but by most plugins by using their FQCN. This is a way to distribute modules, lookups, filters, and so on without having to import a role in every play.

Vars plugins in collections are not loaded automatically, and always require being explicitly enabled by using their fully qualified collection name. See *Enabling vars plugins* for details.

Cache plugins in collections may be used for fact caching, but are not supported for inventory plugins.

module_utils

When coding with `module_utils` in a collection, the Python `import` statement needs to take into account the FQCN along with the `ansible_collections` convention. The resulting Python import will look like `from ansible_collections.{namespace}.{collection}.plugins.module_utils.{util} import {something}`

The following example snippets show a Python and PowerShell module using both default Ansible `module_utils` and those provided by a collection. In this example the namespace is `community`, the collection is `test_collection`. In the Python example the `module_util` in question is called `qradar` such that the FQCN is `community.test_collection.plugins.module_utils.qradar`:

```
from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.common.text.converters import to_text

from ansible.module_utils.six.moves.urllib.parse import urlencode, quote_plus
from ansible.module_utils.six.moves.urllib.error import HTTPError
from ansible_collections.community.test_collection.plugins.module_utils.qradar import␣
↪QRadarRequest

argspec = dict(
    name=dict(required=True, type='str'),
    state=dict(choices=['present', 'absent'], required=True),
)

module = AnsibleModule(
    argument_spec=argspec,
    supports_check_mode=True
)

qradar_request = QRadarRequest(
    module,
    headers={"Content-Type": "application/json"},
    not_rest_data_keys=['state']
)
```

Note that importing something from an `__init__.py` file requires using the file name:

```
from ansible_collections.namespace.collection_name.plugins.callback.__init__ import␣
↪CustomBaseClass
```

In the PowerShell example the `module_util` in question is called `hyperv` such that the FQCN is `community.test_collection.plugins.module_utils.hyperv`:

```
#!powershell
#AnsibleRequires -CSharpUtil Ansible.Basic
#AnsibleRequires -PowerShell ansible_collections.community.test_collection.plugins.
↪module_utils.hyperv

$spec = @{
    name = @{ required = $true; type = "str" }
    state = @{ required = $true; choices = @("present", "absent") }
}
$module = [Ansible.Basic.AnsibleModule]::Create($args, $spec)

Invoke-HyperVFunction -Name $module.Params.name

$module.ExitJson()
```

roles directory

Collection roles are mostly the same as existing roles, but with a couple of limitations:

- Role names are now limited to contain only lowercase alphanumeric characters, plus `_` and start with an alpha character.
- Roles in a collection cannot contain plugins any more. Plugins must live in the collection `plugins` directory tree. Each plugin is accessible to all roles in the collection.

The directory name of the role is used as the role name. Therefore, the directory name must comply with the above role name rules. The collection import into Galaxy will fail if a role name does not comply with these rules.

You can migrate 'traditional roles' into a collection but they must follow the rules above. You may need to rename roles if they don't conform. You will have to move or link any role-based plugins to the collection specific directories.

Note: For roles imported into Galaxy directly from a GitHub repository, setting the `role_name` value in the role's metadata overrides the role name used by Galaxy. For collections, that value is ignored. When importing a collection, Galaxy uses the role directory as the name of the role and ignores the `role_name` metadata value.

playbooks directory

In prior releases, you could reference playbooks in this directory using the full path to the playbook file from the command line. In ansible-core 2.11 and later, you can use the FQCN, `namespace.collection.playbook` (with or without extension), to reference the playbooks from the command line or from `import_playbook`. This will keep the playbook in 'collection context', as if you had added `collections: [ namespace.collection ]` to it.

You can have most of the subdirectories you would expect, such `files/`, `vars/` or `templates/` but no `roles/` since those are handled already in the collection.

Also, playbooks within a collection follow the same guidelines as any playbooks except for these few adjustments:

- Directory: It must be in the `playbooks/` directory.
- Hosts: The host should be defined as a variable so the users of a playbook do not mistakenly run the plays against their entire inventory (if the host is set to all). For example - `hosts: '{{target|default("all")}}'`.

To run the plays, users can now use such commands as `ansible-playbook --e 'targets=webservers'` or `ansible-playbook --limit webservers`. Either way, the collection owner should document their playbooks and how to use them in the `docs/` folder or `README` file.

tests directory

Ansible Collections are tested much like Ansible itself, by using the *ansible-test* utility which is released as part of Ansible, version 2.9.0 and newer. Because Ansible Collections are tested using the same tooling as Ansible itself, by using the *ansible-test*, all Ansible developer documentation for testing is applicable for authoring Collections Tests with one key concept to keep in mind.

See *Testing collections* for specific information on how to test collections with `ansible-test`.

When reading the *Testing Ansible* documentation, there will be content that applies to running Ansible from source code through a git clone, which is typical of an Ansible developer. However, it's not always typical for an Ansible Collection author to be running Ansible from source but instead from a stable release, and to create Collections it is not necessary to run Ansible from source. Therefore, when references of dealing with *ansible-test* binary paths, command completion, or environment variables are presented throughout the *Testing Ansible* documentation; keep in mind that it is not needed for Ansible Collection Testing because the act of installing the stable release of Ansible containing *ansible-test* is expected to setup those things for you.

meta directory and runtime.yml

A collection can store some additional metadata in a `runtime.yml` file in the collection's `meta` directory. The `runtime.yml` file supports the top level keys:

- *requires_ansible*:

  The version of Ansible Core (ansible-core) required to use the collection. Multiple versions can be separated with a comma.

  ```
  requires_ansible: ">=2.10,<2.11"
  ```

  Note: although the version is a PEP440 Version Specifier under the hood, Ansible deviates from PEP440 behavior by truncating prerelease segments from the Ansible version. This means that Ansible 2.11.0b1 is compatible with something that `requires_ansible: ">=2.11"`.

- *plugin_routing*:

  Content in a collection that Ansible needs to load from another location or that has been deprecated/removed. The top level keys of `plugin_routing` are types of plugins, with individual plugin names as subkeys. To define a new location for a plugin, set the `redirect` field to another name. To deprecate a plugin, use the `deprecation` field to provide a custom warning message and the removal version or date. If the plugin has been renamed or moved to a new location, the `redirect` field should also be provided. If a plugin is being removed entirely, `tombstone` can be used for the fatal error message and removal version or date.

  ```
  plugin_routing:
    inventory:
      kubevirt:
        redirect: community.general.kubevirt
      my_inventory:
        tombstone:
          removal_version: "2.0.0"
          warning_text: my_inventory has been removed. Please use other_inventory␣
  ↪instead.
    modules:
      my_module:
        deprecation:
          removal_date: "2021-11-30"
          warning_text: my_module will be removed in a future release of this␣
  ↪collection. Use another.collection.new_module instead.
        redirect: another.collection.new_module
      podman_image:
        redirect: containers.podman.podman_image
    module_utils:
      ec2:
        redirect: amazon.aws.ec2
      util_dir.subdir.my_util:
        redirect: namespace.name.my_util
  ```

- *import_redirection*

A mapping of names for Python import statements and their redirected locations.

```
import_redirection:
  ansible.module_utils.old_utility:
    redirect: ansible_collections.namespace_name.collection_name.plugins.module_
↪utils.new_location
```

- *action_groups*

  A mapping of groups and the list of action plugin and module names they contain. They may also have a special 'metadata' dictionary in the list, which can be used to include actions from other groups.

```
action_groups:
  groupname:
    # The special metadata dictionary. All action/module names should be strings.
    - metadata:
        extend_group:
          - another.collection.groupname
          - another_group
    - my_action
  another_group:
    - my_module
    - another.collection.another_module
```

See also:

Distributing collections
Learn how to package and publish your collection

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

Collection Galaxy metadata structure

A key component of an Ansible collection is the `galaxy.yml` file placed in the root directory of a collection. This file contains the metadata of the collection that is used to generate a collection artifact.

Structure

The `galaxy.yml` file must contain the following keys in valid YAML:

Key	Comment
namespace string / required	The namespace of the collection. This can be a company/brand/organization or product namespace under which all content lives. May only contain alphanumeric lowercase characters and underscores. Namespaces cannot start with underscores or numbers and cannot contain consecutive underscores.
name string / required	The name of the collection. Has the same character restrictions as `namespace`.
version string / required	The version of the collection. Must be compatible with semantic versioning.
readme string / required	The path to the Markdown (.md) readme file. This path is relative to the root of the collection.
authors list / required	A list of the collection's content authors. Can be just the name or in the format 'Full Name <email> (url) @nicks:irc/im.site#channel'.
description string	A short summary description of the collection.
license list	Either a single license or a list of licenses for content inside of a collection. Ansible Galaxy currently only accepts SPDX licenses This key is mutually exclusive with `license_file`.
license_file string	The path to the license file for the collection. This path is relative to the root of the collection. This key is mutually exclusive with `license`.
tags list	A list of tags you want to associate with the collection for indexing/searching. A tag name has the same character requirements as `namespace` and `name`.
dependencies dictionary	Collections that this collection requires to be installed for it to be usable. The key of the dict is the collection label `namespace.name`. The value is a version range specifiers. Multiple version range specifiers can be set and are separated by `,`.
repository string	The URL of the originating SCM repository.
documentation string	The URL to any online docs.
homepage string	The URL to the homepage of the collection/project.
issues string	The URL to the collection issue tracker.
build_ignor list version_added: 2.10	A list of file glob-like patterns used to filter any files or directories that should not be included in the build artifact. A pattern is matched from the relative path of the file or directory of the collection directory. This uses `fnmatch` to match the files or directories. Some directories and files like `galaxy.yml`, `*.pyc`, `*.retry`, and `.git` are always filtered. Mutually exclusive with `manifest`
manifest sentinel version_added: 2.14	A dict controlling use of manifest directives used in building the collection artifact. The key `directives` is a list of MANIFEST.in style directives The key `omit_default_directives` is a boolean that controls whether the default directives are used Mutually exclusive with `build_ignore`

Examples

```
namespace: "namespace_name"
name: "collection_name"
version: "1.0.12"
readme: "README.md"
authors:
    - "Author1"
    - "Author2 (https://author2.example.com)"
    - "Author3 <author3@example.com>"
dependencies:
    "other_namespace.collection1": ">=1.0.0"
    "other_namespace.collection2": ">=2.0.0,<3.0.0"
    "anderson55.my_collection": "*"    # note: "*" selects the highest version available
license:
    - "MIT"
tags:
    - demo
    - collection
repository: "https://www.github.com/my_org/my_collection"
```

See also:

Developing collections
Develop or modify a collection.

Developing modules
Learn about how to write Ansible modules

Using Ansible collections
Learn how to install and use collections.

Mailing List
The development mailing list

irc.libera.chat
#ansible IRC chat channel

For instructions on developing modules, see *Developing modules*.

See also:

Using Ansible collections
Learn how to install and use collections in playbooks and roles

Contributing to Ansible-maintained Collections
Guidelines for contributing to selected collections

Ansible Collections Overview and FAQ
Current development status of community collections and FAQ

Mailing List
The development mailing list

Real-time chat
How to join Ansible chat channels

1.17.21 Migrating Roles to Roles in Collections on Galaxy

You can migrate any existing standalone role into a collection and host the collection on Galaxy. With Ansible collections, you can distribute many roles in a single cohesive unit of re-usable automation. Inside a collection, you can share custom plugins across all roles in the collection instead of duplicating them in each role's `library/`` directory.

You must migrate roles to collections if you want to distribute them as certified Ansible content.

Note: If you want to import your collection to Galaxy, you need a Galaxy namespace.

See *Developing collections* for details on collections.

Comparing standalone roles to collection roles

Standalone roles have the following directory structure:

```
  role/
  ├── defaults
  ├── files
  ├── handlers
  ├── library
  ├── meta
  ├── module_utils
  ├── [*_plugins]
  ├── tasks
  ├── templates
  ├── tests
  └── vars
```

The highlighted directories above will change when you migrate to a collection-based role. The collection directory structure includes a `roles/` directory:

```
mynamespace/
└── mycollection/
  ├── docs/
  ├── galaxy.yml
  ├── plugins/
  │   ├── modules/
  │   │   └── module1.py
  │   ├── inventory/
  │   └── .../
  ├── README.md
  ├── roles/
  │   ├── role1/
```

(continues on next page)

(continued from previous page)

```
    │   ├── role2/
    │   └── .../
    ├── playbooks/
    │   ├── files/
    │   ├── vars/
    │   ├── templates/
    │   └── tasks/
    └── tests/
```

You will need to use the Fully Qualified Collection Name (FQCN) to use the roles and plugins when you migrate your role into a collection. The FQCN is the combination of the collection `namespace`, collection `name`, and the content item you are referring to.

So for example, in the above collection, the FQCN to access `role1` would be:

```
mynamespace.mycollection.role1
```

A collection can contain one or more roles in the `roles/` directory and these are almost identical to standalone roles, except you need to move plugins out of the individual roles, and use the FQCN (Fully Qualified Collection Name) in some places, as detailed in the next section.

Note: In standalone roles, some of the plugin directories referenced their plugin types in the plural sense; this is not the case in collections.

Migrating a role to a collection

To migrate from a standalone role that contains no plugins to a collection role:

1. Create a local `ansible_collections` directory and `cd` to this new directory.
2. Create a collection. If you want to import this collection to Ansible Galaxy, you need a Galaxy namespace.

```
$ ansible-galaxy collection init mynamespace.mycollection
```

This creates the collection directory structure.

3. Copy the standalone role directory into the `roles/` subdirectory of the collection. Roles in collections cannot have hyphens in the role name. Rename any such roles to use underscores instead.

```
$ mkdir mynamespace/mycollection/roles/my_role/
$ cp -r /path/to/standalone/role/mynamespace/my_role/\* mynamespace/mycollection/roles/
↪my_role/
```

4. Update `galaxy.yml` to include any role dependencies.
5. Update the collection README.md file to add links to any role README.md files.

Migrating a role that contains plugins to a collection

To migrate from a standalone role that has plugins to a collection role:

1. Create a local `ansible_collections directory` and `cd` to this new directory.
2. Create a collection. If you want to import this collection to Ansible Galaxy, you need a Galaxy namespace.

```
$ ansible-galaxy collection init mynamespace.mycollection
```

This creates the collection directory structure.

3. Copy the standalone role directory into the `roles/` subdirectory of the collection. Roles in collections cannot have hyphens in the role name. Rename any such roles to use underscores instead.

```
$ mkdir mynamespace/mycollection/roles/my_role/
$ cp -r /path/to/standalone/role/mynamespace/my_role/\* mynamespace/mycollection/roles/
↪my_role/
```

4. Move any modules to the `plugins/modules/` directory.

```
$ mv -r mynamespace/mycollection/roles/my_role/library/\* mynamespace/mycollection/
↪plugins/modules/
```

5. Move any other plugins to the appropriate `plugins/PLUGINTYPE/` directory. See *Migrating other role plugins to a collection* for additional steps that may be required.
6. Update `galaxy.yml` to include any role dependencies.
7. Update the collection README.md file to add links to any role README.md files.
8. Change any references to the role to use the FQCN.

```
---
- name: example role by FQCN
  hosts: some_host_pattern
  tasks:
    - name: import FQCN role from a collection
      import_role:
        name: mynamespace.mycollection.my_role
```

You can alternately use the `collections` keyword to simplify this:

```
---
- name: example role by FQCN
  hosts: some_host_pattern
  collections:
    - mynamespace.mycollection
  tasks:
    - name: import role from a collection
      import_role:
        name: my_role
```

Migrating other role plugins to a collection

To migrate other role plugins to a collection:

1. Move each nonmodule plugins to the appropriate `plugins/PLUGINTYPE/` directory. The `mynamespace/mycollection/plugins/README.md` file explains the types of plugins that the collection can contain within optionally created subdirectories.

```
$ mv -r mynamespace/mycollection/roles/my_role/filter_plugins/\* mynamespace/
↪mycollection/plugins/filter/
```

2. Update documentation to use the FQCN. Plugins that use `doc_fragments` need to use FQCN (for example, `mydocfrag` becomes `mynamespace.mycollection.mydocfrag`).

3. Update relative imports work in collections to start with a period. For example, `./filename` and `../asdfu/filestuff` works but `filename` in same directory must be updated to `./filename`.

If you have a custom `module_utils` or import from `__init__.py`, you must also:

1. Change the Python namespace for custom `module_utils` to use the FQCN along with the `ansible_collections` convention. See *Updating module_utils*.

2. Change how you import from `__init__.py`. See *Importing from __init__.py*.

Updating `module_utils`

If any of your custom modules use a custom module utility, once you migrate to a collection you cannot address the module utility in the top level `ansible.module_utils` Python namespace. Ansible does not merge content from collections into the Ansible internal Python namespace. Update any Python import statements that refer to custom module utilities when you migrate your custom content to collections. See *module_utils in collections* for more details.

When coding with `module_utils` in a collection, the Python import statement needs to take into account the FQCN along with the `ansible_collections` convention. The resulting Python import looks similar to the following example:

```
from ansible_collections.{namespace}.{collectionname}.plugins.module_utils.{util} import
↪{something}
```

Note: You need to follow the same rules in changing paths and using namespaced names for subclassed plugins.

The following example code snippets show a Python and a PowerShell module using both default Ansible `module_utils` and those provided by a collection. In this example the namespace is `ansible_example` and the collection is `community`.

In the Python example the `module_utils` is `helper` and the FQCN is `ansible_example.community.plugins.module_utils.helper`:

```
from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.common.text.converters import to_text
from ansible.module_utils.six.moves.urllib.parse import urlencode
from ansible.module_utils.six.moves.urllib.error import HTTPError
from ansible_collections.ansible_example.community.plugins.module_utils.helper import␣
↪HelperRequest
```

(continues on next page)

(continued from previous page)

```
argspec = dict(
        name=dict(required=True, type='str'),
        state=dict(choices=['present', 'absent'], required=True),
)

module = AnsibleModule(
        argument_spec=argspec,
        supports_check_mode=True
)

_request = HelperRequest(
      module,
        headers={"Content-Type": "application/json"},
     data=data
)
```

In the PowerShell example the `module_utils` is `hyperv` and the FQCN is `ansible_example.community.plugins.module_utils.hyperv`:

```
#!powershell
#AnsibleRequires -CSharpUtil Ansible.Basic
#AnsibleRequires -PowerShell ansible_collections.ansible_example.community.plugins.
↪module_utils.hyperv

$spec = @{
        name = @{ required = $true; type = "str" }
      state = @{ required = $true; choices = @("present", "absent") }
}
$module = [Ansible.Basic.AnsibleModule]::Create($args, $spec)

Invoke-HyperVFunction -Name $module.Params.name

$module.ExitJson()
```

Importing from __init__.py

Because of the way that the CPython interpreter does imports, combined with the way the Ansible plugin loader works, if your custom embedded module or plugin requires importing something from an `__init__.py` file, that also becomes part of your collection. You can either originate the content inside a standalone role or use the file name in the Python import statement. The following example is an `__init__.py` file that is part of a callback plugin found inside a collection named `ansible_example.community`.

```
from ansible_collections.ansible_example.community.plugins.callback.__init__ import␣
↪CustomBaseClass
```

Example: Migrating a standalone role with plugins to a collection

In this example we have a standalone role called `my-standalone-role.webapp` to emulate a standalone role that contains dashes in the name (which is not valid in collections). This standalone role contains a custom module in the `library/` directory called `manage_webserver`.

```
my-standalone-role.webapp
├── defaults
├── files
├── handlers
├── library
├── meta
├── tasks
├── templates
├── tests
└── vars
```

1. Create a new collection, for example, `acme.webserver`:

```
$ ansible-galaxy collection init acme.webserver
- Collection acme.webserver was created successfully
$ tree acme -d 1
acme
└── webserver
       ├── docs
       ├── plugins
       └── roles
```

2. Create the `webapp` role inside the collection and copy all contents from the standalone role:

```
$ mkdir acme/webserver/roles/webapp
$ cp my-standalone-role.webapp/* acme/webserver/roles/webapp/
```

3. Move the `manage_webserver` module to its new home in `acme/webserver/plugins/modules/`:

```
$ cp my-standalone-role.webapp/library/manage_webserver.py acme/webserver/plugins/
↪modules/manage.py
```

Note: This example changed the original source file `manage_webserver.py` to the destination file `manage.py`. This is optional but the FQCN provides the `webserver` context as `acme.webserver.manage`.

4. Change `manage_webserver` to `acme.webserver.manage` in `tasks/` files in the role (for example, `my-standalone-role.webapp/tasks/main.yml`) and any use of the original module name.

Note: This name change is only required if you changed the original module name, but illustrates content referenced by FQCN can offer context and in turn can make module and plugin names shorter. If you anticipate using these modules independent of the role, keep the original naming conventions. Users can add the *collections keyword* in their playbooks. Typically roles are an abstraction layer and users won't use components of the role independently.

Example: Supporting standalone roles and migrated collection roles in a downstream RPM

A standalone role can co-exist with its collection role counterpart (for example, as part of a support lifecycle of a product). This should only be done for a transition period, but these two can exist in downstream in packages such as RPMs. For example, the RHEL system roles could coexist with an example of a RHEL system roles collection and provide existing backwards compatibility with the downstream RPM.

This section walks through an example creating this coexistence in a downstream RPM and requires Ansible 2.9.0 or later.

To deliver a role as both a standalone role and a collection role:

1. Place the collection in `/usr/share/ansible/collections/ansible_collections/`.
2. Copy the contents of the role inside the collection into a directory named after the standalone role and place the standalone role in `/usr/share/ansible/roles/`.

All previously bundled modules and plugins used in the standalone role are now referenced by FQCN so even though they are no longer embedded, they can be found from the collection contents.This is an example of how the content inside the collection is a unique entity and does not have to be bound to a role or otherwise. You could alternately create two separate collections: one for the modules and plugins and another for the standalone role to migrate to. The role must use the modules and plugins as FQCN.

The following is an example RPM spec file that accomplishes this using this example content:

```
Name: acme-ansible-content
Summary: Ansible Collection for deploying and configuring ACME webapp
Version: 1.0.0
Release: 1%{?dist}
License: GPLv3+
Source0: acme-webserver-1.0.0.tar.gz

Url: https://github.com/acme/webserver-ansible-collection
BuildArch: noarch

%global roleprefix my-standalone-role.
%global collection_namespace acme
%global collection_name webserver

%global collection_dir %{_datadir}/ansible/collections/ansible_collections/%{collection_
↪namespace}/%{collection_name}

%description
Ansible Collection and standalone role (for backward compatibility and migration) to␣
↪deploy, configure, and manage the ACME webapp software.

%prep
%setup -qc

%build

%install

mkdir -p %{buildroot}/%{collection_dir}
cp -r ./* %{buildroot}/%{collection_dir}/
```

(continues on next page)

(continued from previous page)

```
mkdir -p %{buildroot}/%{_datadir}/ansible/roles
for role in %{buildroot}/%{collection_dir}/roles/*
  do
         cp -pR ${role} %{buildroot}/%{_datadir}/ansible/roles/%{roleprefix}$(basename $
↪{role})

         mkdir -p %{buildroot}/%{_pkgdocdir}/$(basename ${role})
         for docfile in README.md COPYING LICENSE
          do
      if [ -f ${role}/${docfile} ]
          then
              cp -p ${role}/${docfile} %{buildroot}/%{_pkgdocdir}/$(basename ${role})/$
↪{docfile}
      fi
         done
done


%files
%dir %{_datadir}/ansible
%dir %{_datadir}/ansible/roles
%dir %{_datadir}/ansible/collections
%dir %{_datadir}/ansible/collections/ansible_collections
%{_datadir}/ansible/roles/
%doc %{_pkgdocdir}/*/README.md
%doc %{_datadir}/ansible/roles/%{roleprefix}*/README.md
%{collection_dir}
%doc %{collection_dir}/roles/*/README.md
%license %{_pkgdocdir}/*/COPYING
%license %{_pkgdocdir}/*/LICENSE
```

Using `ansible.legacy` to access local custom modules from collections-based roles

Some roles within a collection use *local custom modules* that are not part of the collection itself. If there is a conflict between the custom module short name and the collection module name, you need to specify which module your tasks call. You can update the tasks to change `local_module_name` to `ansible.legacy.local_module_name` to ensure you are using the custom module.

1.17.22 Ansible architecture

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

Being designed for multi-tier deployments since day one, Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time.

It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

In this section, we'll give you a really quick overview of how Ansible works so you can see how the pieces fit together.

Modules

Ansible works by connecting to your nodes and pushing out scripts called "Ansible modules" to them. Most modules accept parameters that describe the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.

You can *write your own modules*, though you should first consider *whether you should*. Typically you'll work with your favorite terminal program, a text editor, and probably a version control system to keep track of changes to your content. You may write specialized modules in any language that can return JSON (Ruby, Python, bash, and so on).

Module utilities

When multiple modules use the same code, Ansible stores those functions as module utilities to minimize duplication and maintenance. For example, the code that parses URLs is `lib/ansible/module_utils/url.py`. You can *write your own module utilities* as well. Module utilities may only be written in Python or in PowerShell.

Plugins

Plugins augment Ansible's core functionality. While modules execute on the target system in separate processes (usually that means on a remote system), plugins execute on the control node within the `/usr/bin/ansible` process. Plugins offer options and extensions for the core features of Ansible - transforming data, logging output, connecting to inventory, and more. Ansible ships with a number of handy plugins, and you can easily *write your own*. For example, you can write an *inventory plugin* to connect to any datasource that returns JSON. Plugins must be written in Python.

Inventory

By default, Ansible represents the machines it manages in a file (INI, YAML, and so on) that puts all of your managed machines in groups of your own choosing.

To add new machines, there is no additional SSL signing server involved, so there's never any hassle deciding why a particular machine didn't get linked up due to obscure NTP or DNS issues.

If there's another source of truth in your infrastructure, Ansible can also connect to that. Ansible can draw inventory, group, and variable information from sources like EC2, Rackspace, OpenStack, and more.

Here's what a plain text inventory file looks like:

```
---
[webservers]
www1.example.com
www2.example.com
```

(continues on next page)

(continued from previous page)

```

[dbservers]
db0.example.com
db1.example.com
```

Once inventory hosts are listed, variables can be assigned to them in simple text files (in a subdirectory called 'group_vars/' or 'host_vars/') or directly in the inventory file.

Or, as already mentioned, use a dynamic inventory to pull your inventory from data sources like EC2, Rackspace, or OpenStack.

Playbooks

Playbooks can finely orchestrate multiple slices of your infrastructure topology, with very detailed control over how many machines to tackle at a time. This is where Ansible starts to get most interesting.

Ansible's approach to orchestration is one of finely-tuned simplicity, as we believe your automation code should make perfect sense to you years down the road and there should be very little to remember about special syntax or features.

Here's what a simple playbook looks like:

```yaml
---
- hosts: webservers
  serial: 5 # update 5 machines at a time
  roles:
  - common
  - webapp

- hosts: content_servers
  roles:
  - common
  - content
```

The Ansible search path

Modules, module utilities, plugins, playbooks, and roles can live in multiple locations. If you write your own code to extend Ansible's core features, you may have multiple files with similar or the same names in different locations on your Ansible control node. The search path determines which of these files Ansible will discover and use on any given playbook run.

Ansible's search path grows incrementally over a run. As Ansible finds each playbook and role included in a given run, it appends any directories related to that playbook or role to the search path. Those directories remain in scope for the duration of the run, even after the playbook or role has finished executing. Ansible loads modules, module utilities, and plugins in this order:

1. Directories adjacent to a playbook specified on the command line. If you run Ansible with `ansible-playbook /path/to/play.yml`, Ansible appends these directories if they exist:

   ```
   /path/to/modules
   /path/to/module_utils
   /path/to/plugins
   ```

2. Directories adjacent to a playbook that is statically imported by a playbook specified on the command line. If `play.yml` includes `- import_playbook: /path/to/subdir/play1.yml`, Ansible appends these directories if they exist:

   ```
   /path/to/subdir/modules
   /path/to/subdir/module_utils
   /path/to/subdir/plugins
   ```

3. Subdirectories of a role directory referenced by a playbook. If `play.yml` runs `myrole`, Ansible appends these directories if they exist:

   ```
   /path/to/roles/myrole/modules
   /path/to/roles/myrole/module_utils
   /path/to/roles/myrole/plugins
   ```

4. Directories specified as default paths in `ansible.cfg` or by the related environment variables, including the paths for the various plugin types. See *Ansible Configuration Settings* for more information. Sample `ansible.cfg` fields:

   ```
   DEFAULT_MODULE_PATH
   DEFAULT_MODULE_UTILS_PATH
   DEFAULT_CACHE_PLUGIN_PATH
   DEFAULT_FILTER_PLUGIN_PATH
   ```

   Sample environment variables:

   ```
   ANSIBLE_LIBRARY
   ANSIBLE_MODULE_UTILS
   ANSIBLE_CACHE_PLUGINS
   ANSIBLE_FILTER_PLUGINS
   ```

5. The standard directories that ship as part of the Ansible distribution.

Caution: Modules, module utilities, and plugins in user-specified directories will override the standard versions. This includes some files with generic names. For example, if you have a file named `basic.py` in a user-specified directory, it will override the standard `ansible.module_utils.basic`.

If you have more than one module, module utility, or plugin with the same name in different user-specified directories, the order of commands at the command line and the order of includes and roles in each play will affect which one is found and used on that particular play.

1.18 Legacy Public Cloud Guides

The legacy guides in this section may be out of date. They cover using Ansible with a range of public cloud platforms. They explore particular use cases in greater depth and provide a more "top-down" explanation of some basic features.

Guides for using public clouds are moving into collections. We are migrating these guides into collections. Please update your links for the following guides:

Amazon Web Services Guide

1.18.1 Alibaba Cloud Compute Services Guide

Introduction

Ansible contains several modules for controlling and managing Alibaba Cloud Compute Services (Alicloud). This guide explains how to use the Alicloud Ansible modules together.

All Alicloud modules require `footmark` - install it on your control machine with `pip install footmark`.

Cloud modules, including Alicloud modules, execute on your local machine (the control machine) with `connection: local`, rather than on remote machines defined in your hosts.

Normally, you'll use the following pattern for plays that provision Alicloud resources:

```
- hosts: localhost
  connection: local
  vars:
    - ...
  tasks:
    - ...
```

Authentication

You can specify your Alicloud authentication credentials (access key and secret key) by passing them as environment variables or by storing them in a vars file.

To pass authentication credentials as environment variables:

```
export ALICLOUD_ACCESS_KEY='Alicloud123'
export ALICLOUD_SECRET_KEY='AlicloudSecret123'
```

To store authentication credentials in a vars_files, encrypt them with *Ansible Vault* to keep them secure, then list them:

```
---
alicloud_access_key: "--REMOVED--"
alicloud_secret_key: "--REMOVED--"
```

Note that if you store your credentials in a vars_files, you need to refer to them in each Alicloud module. For example:

```
- ali_instance:
    alicloud_access_key: "{{alicloud_access_key}}"
    alicloud_secret_key: "{{alicloud_secret_key}}"
    image_id: "..."
```

Provisioning

Alicloud modules create Alicloud ECS instances, disks, virtual private clouds, virtual switches, security groups and other resources.

You can use the `count` parameter to control the number of resources you create or terminate. For example, if you want exactly 5 instances tagged `NewECS`, set the `count` of instances to 5 and the `count_tag` to `NewECS`, as shown in the last task of the example playbook below. If there are no instances with the tag `NewECS`, the task creates 5 new instances. If there are 2 instances with that tag, the task creates 3 more. If there are 8 instances with that tag, the task terminates 3 of those instances.

If you do not specify a `count_tag`, the task creates the number of instances you specify in `count` with the `instance_name` you provide.

```
# alicloud_setup.yml

- hosts: localhost
  connection: local

  tasks:

    - name: Create VPC
      ali_vpc:
        cidr_block: '{{ cidr_block }}'
        vpc_name: new_vpc
      register: created_vpc

    - name: Create VSwitch
      ali_vswitch:
        alicloud_zone: '{{ alicloud_zone }}'
        cidr_block: '{{ vsw_cidr }}'
        vswitch_name: new_vswitch
        vpc_id: '{{ created_vpc.vpc.id }}'
      register: created_vsw

    - name: Create security group
      ali_security_group:
        name: new_group
        vpc_id: '{{ created_vpc.vpc.id }}'
        rules:
          - proto: tcp
            port_range: 22/22
            cidr_ip: 0.0.0.0/0
            priority: 1
        rules_egress:
          - proto: tcp
            port_range: 80/80
            cidr_ip: 192.168.0.54/32
            priority: 1
      register: created_group

    - name: Create a set of instances
      ali_instance:
         security_groups: '{{ created_group.group_id }}'
         instance_type: ecs.n4.small
         image_id: "{{ ami_id }}"
         instance_name: "My-new-instance"
         instance_tags:
             Name: NewECS
             Version: 0.0.1
         count: 5
         count_tag:
             Name: NewECS
         allocate_public_ip: true
```

(continues on next page)

(continued from previous page)

```
          max_bandwidth_out: 50
          vswitch_id: '{{ created_vsw.vswitch.id}}'
        register: create_instance
```

In the example playbook above, data about the vpc, vswitch, group, and instances created by this playbook are saved in the variables defined by the "register" keyword in each task.

Each Alicloud module offers a variety of parameter options. Not all options are demonstrated in the above example. See each individual module for further details and examples.

1.18.2 CloudStack Cloud Guide

Introduction

The purpose of this section is to explain how to put Ansible modules together to use Ansible in a CloudStack context. You will find more usage examples in the details section of each module.

Ansible contains a number of extra modules for interacting with CloudStack based clouds. All modules support check mode, are designed to be idempotent, have been created and tested, and are maintained by the community.

Note: Some of the modules will require domain admin or root admin privileges.

Prerequisites

Prerequisites for using the CloudStack modules are minimal. In addition to Ansible itself, all of the modules require the python library `cs` https://pypi.org/project/cs/

You'll need this Python module installed on the execution host, usually your workstation.

```
$ pip install cs
```

Or alternatively starting with Debian 9 and Ubuntu 16.04:

```
$ sudo apt install python-cs
```

Note: cs also includes a command line interface for ad hoc interaction with the CloudStack API, for example `$ cs listVirtualMachines state=Running`.

Limitations and Known Issues

VPC support has been improved since Ansible 2.3 but is still not yet fully implemented. The community is working on the VPC integration.

Credentials File

You can pass credentials and the endpoint of your cloud as module arguments, however in most cases it is a far less work to store your credentials in the cloudstack.ini file.

The python library cs looks for the credentials file in the following order (last one wins):

- A `.cloudstack.ini` (note the dot) file in the home directory.
- A `CLOUDSTACK_CONFIG` environment variable pointing to an .ini file.
- A `cloudstack.ini` (without the dot) file in the current working directory, same directory as your playbooks are located.

The structure of the ini file must look like this:

```
$ cat $HOME/.cloudstack.ini
[cloudstack]
endpoint = https://cloud.example.com/client/api
key = api key
secret = api secret
timeout = 30
```

Note: The section `[cloudstack]` is the default section. `CLOUDSTACK_REGION` environment variable can be used to define the default section.

New in version 2.4.

The ENV variables support `CLOUDSTACK_*` as written in the documentation of the library `cs`, like `CLOUDSTACK_TIMEOUT`, `CLOUDSTACK_METHOD`, and so on. has been implemented into Ansible. It is even possible to have some incomplete config in your cloudstack.ini:

```
$ cat $HOME/.cloudstack.ini
[cloudstack]
endpoint = https://cloud.example.com/client/api
timeout = 30
```

and fulfill the missing data by either setting ENV variables or tasks params:

```
---
- name: provision our VMs
  hosts: cloud-vm
  tasks:
    - name: ensure VMs are created and running
      delegate_to: localhost
      cs_instance:
        api_key: your api key
        api_secret: your api secret
        ...
```

Regions

If you use more than one CloudStack region, you can define as many sections as you want and name them as you like, for example:

```
$ cat $HOME/.cloudstack.ini
[exoscale]
endpoint = https://api.exoscale.ch/compute
key = api key
secret = api secret

[example_cloud_one]
endpoint = https://cloud-one.example.com/client/api
key = api key
secret = api secret

[example_cloud_two]
endpoint = https://cloud-two.example.com/client/api
key = api key
secret = api secret
```

Hint: Sections can also be used to for login into the same region using different accounts.

By passing the argument `api_region` with the CloudStack modules, the region wanted will be selected.

```
- name: ensure my ssh public key exists on Exoscale
  cs_sshkeypair:
    name: my-ssh-key
    public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
    api_region: exoscale
  delegate_to: localhost
```

Or by looping over a regions list if you want to do the task in every region:

```
- name: ensure my ssh public key exists in all CloudStack regions
  local_action: cs_sshkeypair
    name: my-ssh-key
    public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
    api_region: "{{ item }}"
    loop:
      - exoscale
      - example_cloud_one
      - example_cloud_two
```

Environment Variables

New in version 2.3.

Since Ansible 2.3 it is possible to use environment variables for domain (`CLOUDSTACK_DOMAIN`), account (`CLOUDSTACK_ACCOUNT`), project (`CLOUDSTACK_PROJECT`), VPC (`CLOUDSTACK_VPC`) and zone (`CLOUDSTACK_ZONE`). This simplifies the tasks by not repeating the arguments for every tasks.

Below you see an example how it can be used in combination with Ansible's block feature:

```
- hosts: cloud-vm
  tasks:
    - block:
        - name: ensure my ssh public key
          cs_sshkeypair:
            name: my-ssh-key
            public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"

        - name: ensure my ssh public key
          cs_instance:
              display_name: "{{ inventory_hostname_short }}"
              template: Linux Debian 7 64-bit 20GB Disk
              service_offering: "{{ cs_offering }}"
              ssh_key: my-ssh-key
              state: running

      delegate_to: localhost
      environment:
        CLOUDSTACK_DOMAIN: root/customers
        CLOUDSTACK_PROJECT: web-app
        CLOUDSTACK_ZONE: sf-1
```

Note: You are still able overwrite the environment variables using the module arguments, for example `zone: sf-2`

Note: Unlike `CLOUDSTACK_REGION` these additional environment variables are ignored in the CLI `cs`.

Use Cases

The following should give you some ideas how to use the modules to provision VMs to the cloud. As always, there isn't only one way to do it. But as always: keep it simple for the beginning is always a good start.

Use Case: Provisioning in a Advanced Networking CloudStack setup

Our CloudStack cloud has an advanced networking setup, we would like to provision web servers, which get a static NAT and open firewall ports 80 and 443. Further we provision database servers, to which we do not give any access to. For accessing the VMs by SSH we use a SSH jump host.

This is how our inventory looks like:

```
[cloud-vm:children]
webserver
db-server
jumphost

[webserver]
web-01.example.com  public_ip=198.51.100.20
web-02.example.com  public_ip=198.51.100.21

[db-server]
db-01.example.com
db-02.example.com

[jumphost]
jump.example.com  public_ip=198.51.100.22
```

As you can see, the public IPs for our web servers and jumphost has been assigned as variable `public_ip` directly in the inventory.

The configure the jumphost, web servers and database servers, we use `group_vars`. The `group_vars` directory contains 4 files for configuration of the groups: cloud-vm, jumphost, webserver and db-server. The cloud-vm is there for specifying the defaults of our cloud infrastructure.

```
# file: group_vars/cloud-vm
---
cs_offering: Small
cs_firewall: []
```

Our database servers should get more CPU and RAM, so we define to use a `Large` offering for them.

```
# file: group_vars/db-server
---
cs_offering: Large
```

The web servers should get a `Small` offering as we would scale them horizontally, which is also our default offering. We also ensure the known web ports are opened for the world.

```
# file: group_vars/webserver
---
cs_firewall:
```

(continues on next page)

(continued from previous page)

```
  - { port: 80 }
  - { port: 443 }
```

Further we provision a jump host which has only port 22 opened for accessing the VMs from our office IPv4 network.

```
# file: group_vars/jumphost
---
cs_firewall:
  - { port: 22, cidr: "17.17.17.0/24" }
```

Now to the fun part. We create a playbook to create our infrastructure we call it `infra.yml`:

```
# file: infra.yaml
---
- name: provision our VMs
  hosts: cloud-vm
  tasks:
    - name: run all enclosed tasks from localhost
      delegate_to: localhost
      block:
        - name: ensure VMs are created and running
          cs_instance:
            name: "{{ inventory_hostname_short }}"
            template: Linux Debian 7 64-bit 20GB Disk
            service_offering: "{{ cs_offering }}"
            state: running

        - name: ensure firewall ports opened
          cs_firewall:
            ip_address: "{{ public_ip }}"
            port: "{{ item.port }}"
            cidr: "{{ item.cidr | default('0.0.0.0/0') }}"
          loop: "{{ cs_firewall }}"
          when: public_ip is defined

        - name: ensure static NATs
          cs_staticnat: vm="{{ inventory_hostname_short }}" ip_address="{{ public_ip }}"
          when: public_ip is defined
```

In the above play we defined 3 tasks and use the group `cloud-vm` as target to handle all VMs in the cloud but instead SSH to these VMs, we use `delegate_to:  localhost` to execute the API calls locally from our workstation.

In the first task, we ensure we have a running VM created with the Debian template. If the VM is already created but stopped, it would just start it. If you like to change the offering on an existing VM, you must add `force:  yes` to the task, which would stop the VM, change the offering and start the VM again.

In the second task we ensure the ports are opened if we give a public IP to the VM.

In the third task we add static NAT to the VMs having a public IP defined.

Note: The public IP addresses must have been acquired in advance, also see `cs_ip_address`

Note: For some modules, for example `cs_sshkeypair` you usually want this to be executed only once, not for every

VM. Therefore you would make a separate play for it targeting localhost. You find an example in the use cases below.

Use Case: Provisioning on a Basic Networking CloudStack setup

A basic networking CloudStack setup is slightly different: Every VM gets a public IP directly assigned and security groups are used for access restriction policy.

This is how our inventory looks like:

```
[cloud-vm:children]
webserver

[webserver]
web-01.example.com
web-02.example.com
```

The default for your VMs looks like this:

```yaml
# file: group_vars/cloud-vm
---
cs_offering: Small
cs_securitygroups: [ 'default']
```

Our webserver will also be in security group `web`:

```yaml
# file: group_vars/webserver
---
cs_securitygroups: [ 'default', 'web' ]
```

The playbook looks like the following:

```yaml
# file: infra.yaml
---
- name: cloud base setup
  hosts: localhost
  tasks:
  - name: upload ssh public key
    cs_sshkeypair:
      name: defaultkey
      public_key: "{{ lookup('file', '~/.ssh/id_rsa.pub') }}"

  - name: ensure security groups exist
    cs_securitygroup:
      name: "{{ item }}"
    loop:
      - default
      - web

  - name: add inbound SSH to security group default
    cs_securitygroup_rule:
      security_group: default
      start_port: "{{ item }}"
      end_port: "{{ item }}"
```

(continues on next page)

(continued from previous page)

```
    loop:
      - 22

  - name: add inbound TCP rules to security group web
    cs_securitygroup_rule:
      security_group: web
      start_port: "{{ item }}"
      end_port: "{{ item }}"
    loop:
      - 80
      - 443

- name: install VMs in the cloud
  hosts: cloud-vm
  tasks:
  - delegate_to: localhost
    block:
    - name: create and run VMs on CloudStack
      cs_instance:
        name: "{{ inventory_hostname_short }}"
        template: Linux Debian 7 64-bit 20GB Disk
        service_offering: "{{ cs_offering }}"
        security_groups: "{{ cs_securitygroups }}"
        ssh_key: defaultkey
        state: Running
      register: vm

    - name: show VM IP
      debug: msg="VM {{ inventory_hostname }} {{ vm.default_ip }}"

    - name: assign IP to the inventory
      set_fact: ansible_ssh_host={{ vm.default_ip }}

    - name: waiting for SSH to come up
      wait_for: port=22 host={{ vm.default_ip }} delay=5
```

In the first play we setup the security groups, in the second play the VMs will created be assigned to these groups. Further you see, that we assign the public IP returned from the modules to the host inventory. This is needed as we do not know the IPs we will get in advance. In a next step you would configure the DNS servers with these IPs for accessing the VMs with their DNS name.

In the last task we wait for SSH to be accessible, so any later play would be able to access the VM by SSH without failure.

1.18.3 Google Cloud Platform Guide

Introduction

Ansible + Google have been working together on a set of auto-generated Ansible modules designed to consistently and comprehensively cover the entirety of the Google Cloud Platform (GCP).

Ansible contains modules for managing Google Cloud Platform resources, including creating instances, controlling network access, working with persistent disks, managing load balancers, and a lot more.

These new modules can be found under a new consistent name scheme "gcp_*" (Note: gcp_target_proxy and gcp_url_map are legacy modules, despite the "gcp_*" name. Please use gcp_compute_target_proxy and gcp_compute_url_map instead).

Additionally, the gcp_compute inventory plugin can discover all Google Compute Engine (GCE) instances and make them automatically available in your Ansible inventory.

You may see a collection of other GCP modules that do not conform to this naming convention. These are the original modules primarily developed by the Ansible community. You will find some overlapping functionality such as with the "gce" module and the new "gcp_compute_instance" module. Either can be used, but you may experience issues trying to use them together.

While the community GCP modules are not going away, Google is investing effort into the new "gcp_*" modules. Google is committed to ensuring the Ansible community has a great experience with GCP and therefore recommends adopting these new modules if possible.

Requisites

The GCP modules require both the `requests` and the `google-auth` libraries to be installed.

```
$ pip install requests google-auth
```

Alternatively for RHEL / CentOS, the `python-requests` package is also available to satisfy `requests` libraries.

```
$ yum install python-requests
```

Credentials

It's easy to create a GCP account with credentials for Ansible. You have multiple options to get your credentials - here are two of the most common options:

- Service Accounts (Recommended): Use JSON service accounts with specific permissions.
- Machine Accounts: Use the permissions associated with the GCP Instance you're using Ansible on.

For the following examples, we'll be using service account credentials.

To work with the GCP modules, you'll first need to get some credentials in the JSON format:

1. Create a Service Account
2. Download JSON credentials

Once you have your credentials, there are two different ways to provide them to Ansible:

- by specifying them directly as module parameters
- by setting environment variables

Providing Credentials as Module Parameters

For the GCE modules you can specify the credentials as arguments:

- `auth_kind`: type of authentication being used (choices: machineaccount, serviceaccount, application)
- `service_account_email`: email associated with the project
- `service_account_file`: path to the JSON credentials file
- `project`: id of the project
- `scopes`: The specific scopes that you want the actions to use.

For example, to create a new IP address using the `gcp_compute_address` module, you can use the following configuration:

```yaml
- name: Create IP address
  hosts: localhost
  gather_facts: false

  vars:
    service_account_file: /home/my_account.json
    project: my-project
    auth_kind: serviceaccount
    scopes:
      - https://www.googleapis.com/auth/compute

  tasks:

   - name: Allocate an IP Address
     gcp_compute_address:
         state: present
         name: 'test-address1'
         region: 'us-west1'
         project: "{{ project }}"
         auth_kind: "{{ auth_kind }}"
         service_account_file: "{{ service_account_file }}"
         scopes: "{{ scopes }}"
```

Providing Credentials as Environment Variables

Set the following environment variables before running Ansible in order to configure your credentials:

```
GCP_AUTH_KIND
GCP_SERVICE_ACCOUNT_EMAIL
GCP_SERVICE_ACCOUNT_FILE
GCP_SCOPES
```

GCE Dynamic Inventory

The best way to interact with your hosts is to use the gcp_compute inventory plugin, which dynamically queries GCE and tells Ansible what nodes can be managed.

To be able to use this GCE dynamic inventory plugin, you need to enable it first by specifying the following in the `ansible.cfg` file:

```
[inventory]
enable_plugins = gcp_compute
```

Then, create a file that ends in `.gcp.yml` in your root directory.

The gcp_compute inventory script takes in the same authentication information as any module.

Here's an example of a valid inventory file:

```
plugin: gcp_compute
projects:
  - graphite-playground
auth_kind: serviceaccount
service_account_file: /home/alexstephen/my_account.json
```

Executing `ansible-inventory --list -i <filename>.gcp.yml` will create a list of GCP instances that are ready to be configured using Ansible.

Create an instance

The full range of GCP modules provide the ability to create a wide variety of GCP resources with the full support of the entire GCP API.

The following playbook creates a GCE Instance. This instance relies on other GCP resources like Disk. By creating other resources separately, we can give as much detail as necessary about how we want to configure the other resources, for example formatting of the Disk. By registering it to a variable, we can simply insert the variable into the instance task. The gcp_compute_instance module will figure out the rest.

```
- name: Create an instance
  hosts: localhost
  gather_facts: false
  vars:
      gcp_project: my-project
      gcp_cred_kind: serviceaccount
      gcp_cred_file: /home/my_account.json
      zone: "us-central1-a"
      region: "us-central1"

  tasks:
   - name: create a disk
     gcp_compute_disk:
         name: 'disk-instance'
         size_gb: 50
         source_image: 'projects/ubuntu-os-cloud/global/images/family/ubuntu-1604-lts'
         zone: "{{ zone }}"
         project: "{{ gcp_project }}"
         auth_kind: "{{ gcp_cred_kind }}"
```

(continues on next page)

(continued from previous page)

```
          service_account_file: "{{ gcp_cred_file }}"
          scopes:
            - https://www.googleapis.com/auth/compute
          state: present
      register: disk
    - name: create a address
      gcp_compute_address:
          name: 'address-instance'
          region: "{{ region }}"
          project: "{{ gcp_project }}"
          auth_kind: "{{ gcp_cred_kind }}"
          service_account_file: "{{ gcp_cred_file }}"
          scopes:
            - https://www.googleapis.com/auth/compute
          state: present
      register: address
    - name: create a instance
      gcp_compute_instance:
          state: present
          name: test-vm
          machine_type: n1-standard-1
          disks:
            - auto_delete: true
              boot: true
              source: "{{ disk }}"
          network_interfaces:
              - network: null # use default
                access_configs:
                  - name: 'External NAT'
                    nat_ip: "{{ address }}"
                    type: 'ONE_TO_ONE_NAT'
          zone: "{{ zone }}"
          project: "{{ gcp_project }}"
          auth_kind: "{{ gcp_cred_kind }}"
          service_account_file: "{{ gcp_cred_file }}"
          scopes:
            - https://www.googleapis.com/auth/compute
      register: instance

    - name: Wait for SSH to come up
      wait_for: host={{ address.address }} port=22 delay=10 timeout=60

    - name: Add host to groupname
      add_host: hostname={{ address.address }} groupname=new_instances


- name: Manage new instances
  hosts: new_instances
  connection: ssh
  become: True
  roles:
    - base_configuration
```

(continues on next page)

(continued from previous page)

```
    - production_server
```

Note that use of the "add_host" module above creates a temporary, in-memory group. This means that a play in the same playbook can then manage machines in the 'new_instances' group, if so desired. Any sort of arbitrary configuration is possible at this point.

For more information about Google Cloud, please visit the Google Cloud website.

Migration Guides

gce.py -> gcp_compute_instance.py

As of Ansible 2.8, we're encouraging everyone to move from the `gce` module to the `gcp_compute_instance` module. The `gcp_compute_instance` module has better support for all of GCP's features, fewer dependencies, more flexibility, and better supports GCP's authentication systems.

The `gcp_compute_instance` module supports all of the features of the `gce` module (and more!). Below is a mapping of `gce` fields over to `gcp_compute_instance` fields.

gce.py	gcp_compute	Notes
state	state/status	State on gce has multiple values: "present", "absent", "stopped", "started", "terminated". State on gcp_compute_instance is used to describe if the instance exists (present) or does not (absent). Status is used to describe if the instance is "started", "stopped" or "terminated".
image	disks[].initializ	You'll need to create a single disk using the disks[] parameter and set it to be the boot disk (disks[].boot = true)
image_family	disks[].initializ	See above.
external_projec	disks[].initializ	The name of the source_image will include the name of the project.
instance_nan	Use a loop or multiple tasks.	Using loops is a more Ansible-centric way of creating multiple instances and gives you the most flexibility.
service_accou	service_accounts[	This is the service_account email address that you want the instance to be associated with. It is not the service_account email address that is used for the credentials necessary to create the instance.
service_accou	service_accounts[	These are the permissions you want to grant to the instance.
pem_file	Not supported.	We recommend using JSON service account credentials instead of PEM files.
credentials_file	service_account_f	
project_id	project	
name	name	This field does not accept an array of names. Use a loop to create multiple instances.
num_instai	Use a loop	For maximum flexibility, we're encouraging users to use Ansible features to create multiple instances, rather than letting the module do it for you.
network	network_interface	
subnetwork	network_interface	
persistent_boot_	disks[].type = 'PERSISTENT'	
disks	disks[]	
ip_forward	can_ip_forwarc	
external_ip	network_interface	This field takes multiple types of values. You can create an IP address with `gcp_compute_address` and place the name/output of the address here. You can also place the string value of the IP address's GCP name or the actual IP address.
disks_auto	disks[].auto_de	
preemptible	scheduling.preemptibl	
disk_size	disks[].initializ	

An example playbook is below:

```yaml
gcp_compute_instance:
    name: "{{ item }}"
    machine_type: n1-standard-1
    ... # any other settings
    zone: us-central1-a
    project: "my-project"
```

(continues on next page)

(continued from previous page)

```
    auth_kind: "service_account_file"
    service_account_file: "~/my_account.json"
    state: present
loop:
  - instance-1
  - instance-2
```

1.18.4 Microsoft Azure Guide

Important: Red Hat Ansible Automation Platform will soon be available on Microsoft Azure. Sign up to preview the experience.

Ansible includes a suite of modules for interacting with Azure Resource Manager, giving you the tools to easily create and orchestrate infrastructure on the Microsoft Azure Cloud.

Requirements

Using the Azure Resource Manager modules requires having specific Azure SDK modules installed on the host running Ansible.

```
$ pip install 'ansible[azure]'
```

If you are running Ansible from source, you can install the dependencies from the root directory of the Ansible repo.

```
$ pip install .[azure]
```

You can also directly run Ansible in Azure Cloud Shell, where Ansible is pre-installed.

Authenticating with Azure

Using the Azure Resource Manager modules requires authenticating with the Azure API. You can choose from two authentication strategies:

- Active Directory Username/Password
- Service Principal Credentials

Follow the directions for the strategy you wish to use, then proceed to *Providing Credentials to Azure Modules* for instructions on how to actually use the modules and authenticate with the Azure API.

Using Service Principal

There is now a detailed official tutorial describing how to create a service principal.

After stepping through the tutorial you will have:

- Your Client ID, which is found in the “client id” box in the “Configure” page of your application in the Azure portal
- Your Secret key, generated when you created the application. You cannot show the key after creation. If you lost the key, you must create a new one in the “Configure” page of your application.

- And finally, a tenant ID. It's a UUID (for example, ABCDEFGH-1234-ABCD-1234-ABCDEFGHIJKL) pointing to the AD containing your application. You will find it in the URL from within the Azure portal, or in the "view endpoints" of any given URL.

Using Active Directory Username/Password

To create an Active Directory username/password:

- Connect to the Azure Classic Portal with your admin account
- Create a user in your default AAD. You must NOT activate Multi-Factor Authentication
- Go to Settings - Administrators
- Click on Add and enter the email of the new user.
- Check the checkbox of the subscription you want to test with this user.
- Login to Azure Portal with this new user to change the temporary password to a new one. You will not be able to use the temporary password for OAuth login.

Providing Credentials to Azure Modules

The modules offer several ways to provide your credentials. For a CI/CD tool such as Ansible AWX or Jenkins, you will most likely want to use environment variables. For local development you may wish to store your credentials in a file within your home directory. And of course, you can always pass credentials as parameters to a task within a playbook. The order of precedence is parameters, then environment variables, and finally a file found in your home directory.

Using Environment Variables

To pass service principal credentials through the environment, define the following variables:

- AZURE_CLIENT_ID
- AZURE_SECRET
- AZURE_SUBSCRIPTION_ID
- AZURE_TENANT

To pass Active Directory username/password through the environment, define the following variables:

- AZURE_AD_USER
- AZURE_PASSWORD
- AZURE_SUBSCRIPTION_ID

To pass Active Directory username/password in ADFS through the environment, define the following variables:

- AZURE_AD_USER
- AZURE_PASSWORD
- AZURE_CLIENT_ID
- AZURE_TENANT
- AZURE_ADFS_AUTHORITY_URL

"AZURE_ADFS_AUTHORITY_URL" is optional. It's necessary only when you have own ADFS authority like https://yourdomain.com/adfs.

Storing in a File

When working in a development environment, it may be desirable to store credentials in a file. The modules will look for credentials in `$HOME/.azure/credentials`. This file is an ini style file. It will look as follows:

```
[default]
subscription_id=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
client_id=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
secret=xxxxxxxxxxxxxxxxx
tenant=xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Note: If your secret values contain non-ASCII characters, you must URL Encode them to avoid login errors.

It is possible to store multiple sets of credentials within the credentials file by creating multiple sections. Each section is considered a profile. The modules look for the [default] profile automatically. Define AZURE_PROFILE in the environment or pass a profile parameter to specify a specific profile.

Passing as Parameters

If you wish to pass credentials as parameters to a task, use the following parameters for service principal:

- client_id
- secret
- subscription_id
- tenant

Or, pass the following parameters for Active Directory username/password:

- ad_user
- password
- subscription_id

Or, pass the following parameters for ADFS username/password:

- ad_user
- password
- client_id
- tenant
- adfs_authority_url

"adfs_authority_url" is optional. It's necessary only when you have own ADFS authority like https://yourdomain.com/adfs.

Other Cloud Environments

To use an Azure Cloud other than the default public cloud (for example, Azure China Cloud, Azure US Government Cloud, Azure Stack), pass the "cloud_environment" argument to modules, configure it in a credential profile, or set the "AZURE_CLOUD_ENVIRONMENT" environment variable. The value is either a cloud name as defined by the Azure Python SDK (for example, "AzureChinaCloud", "AzureUSGovernment"; defaults to "AzureCloud") or an Azure metadata discovery URL (for Azure Stack).

Creating Virtual Machines

There are two ways to create a virtual machine, both involving the azure_rm_virtualmachine module. We can either create a storage account, network interface, security group and public IP address and pass the names of these objects to the module as parameters, or we can let the module do the work for us and accept the defaults it chooses.

Creating Individual Components

An Azure module is available to help you create a storage account, virtual network, subnet, network interface, security group and public IP. Here is a full example of creating each of these and passing the names to the `azure.azcollection.azure_rm_virtualmachine` module at the end:

```
- name: Create storage account
  azure.azcollection.azure_rm_storageaccount:
    resource_group: Testing
    name: testaccount001
    account_type: Standard_LRS

- name: Create virtual network
  azure.azcollection.azure_rm_virtualnetwork:
    resource_group: Testing
    name: testvn001
    address_prefixes: "10.10.0.0/16"

- name: Add subnet
  azure.azcollection.azure_rm_subnet:
    resource_group: Testing
    name: subnet001
    address_prefix: "10.10.0.0/24"
    virtual_network: testvn001

- name: Create public ip
  azure.azcollection.azure_rm_publicipaddress:
    resource_group: Testing
    allocation_method: Static
    name: publicip001

- name: Create security group that allows SSH
  azure.azcollection.azure_rm_securitygroup:
    resource_group: Testing
    name: secgroup001
    rules:
      - name: SSH
        protocol: Tcp
```

(continues on next page)

(continued from previous page)

```
        destination_port_range: 22
        access: Allow
        priority: 101
        direction: Inbound

- name: Create NIC
  azure.azcollection.azure_rm_networkinterface:
    resource_group: Testing
    name: testnic001
    virtual_network: testvn001
    subnet: subnet001
    public_ip_name: publicip001
    security_group: secgroup001

- name: Create virtual machine
  azure.azcollection.azure_rm_virtualmachine:
    resource_group: Testing
    name: testvm001
    vm_size: Standard_D1
    storage_account: testaccount001
    storage_container: testvm001
    storage_blob: testvm001.vhd
    admin_username: admin
    admin_password: Password!
    network_interfaces: testnic001
    image:
      offer: CentOS
      publisher: OpenLogic
      sku: '7.1'
      version: latest
```

Each of the Azure modules offers a variety of parameter options. Not all options are demonstrated in the above example. See each individual module for further details and examples.

Creating a Virtual Machine with Default Options

If you simply want to create a virtual machine without specifying all the details, you can do that as well. The only caveat is that you will need a virtual network with one subnet already in your resource group. Assuming you have a virtual network already with an existing subnet, you can run the following to create a VM:

```
azure.azcollection.azure_rm_virtualmachine:
  resource_group: Testing
  name: testvm10
  vm_size: Standard_D1
  admin_username: chouseknecht
  ssh_password_enabled: false
  ssh_public_keys: "{{ ssh_keys }}"
  image:
    offer: CentOS
    publisher: OpenLogic
    sku: '7.1'
```

(continues on next page)

(continued from previous page)

```
    version: latest
```

Creating a Virtual Machine in Availability Zones

If you want to create a VM in an availability zone, consider the following:

- Both OS disk and data disk must be a 'managed disk', not an 'unmanaged disk'.
- When creating a VM with the `azure.azcollection.azure_rm_virtualmachine` module, you need to explicitly set the `managed_disk_type` parameter to change the OS disk to a managed disk. Otherwise, the OS disk becomes an unmanaged disk.
- When you create a data disk with the `azure.azcollection.azure_rm_manageddisk` module, you need to explicitly specify the `storage_account_type` parameter to make it a managed disk. Otherwise, the data disk will be an unmanaged disk.
- A managed disk does not require a storage account or a storage container, unlike an unmanaged disk. In particular, note that once a VM is created on an unmanaged disk, an unnecessary storage container named "vhds" is automatically created.
- When you create an IP address with the `azure.azcollection.azure_rm_publicipaddress` module, you must set the `sku` parameter to `standard`. Otherwise, the IP address cannot be used in an availability zone.

Dynamic Inventory Script

If you are not familiar with Ansible's dynamic inventory scripts, check out *Intro to Dynamic Inventory*.

The Azure Resource Manager inventory script is called azure_rm.py. It authenticates with the Azure API exactly the same as the Azure modules, which means you will either define the same environment variables described above in *Using Environment Variables*, create a `$HOME/.azure/credentials` file (also described above in *Storing in a File*), or pass command line parameters. To see available command line options execute the following:

```
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/
↪inventory/azure_rm.py
$ ./azure_rm.py --help
```

As with all dynamic inventory scripts, the script can be executed directly, passed as a parameter to the ansible command, or passed directly to ansible-playbook using the -i option. No matter how it is executed the script produces JSON representing all of the hosts found in your Azure subscription. You can narrow this down to just hosts found in a specific set of Azure resource groups, or even down to a specific host.

For a given host, the inventory script provides the following host variables:

```
{
  "ansible_host": "XXX.XXX.XXX.XXX",
  "computer_name": "computer_name2",
  "fqdn": null,
  "id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/providers/
↪Microsoft.Compute/virtualMachines/object-name",
  "image": {
    "offer": "CentOS",
    "publisher": "OpenLogic",
    "sku": "7.1",
    "version": "latest"
```

(continues on next page)

(continued from previous page)

```
  },
  "location": "westus",
  "mac_address": "00-00-5E-00-53-FE",
  "name": "object-name",
  "network_interface": "interface-name",
  "network_interface_id": "/subscriptions/subscription-id/resourceGroups/galaxy-
↪production/providers/Microsoft.Network/networkInterfaces/object-name1",
  "network_security_group": null,
  "network_security_group_id": null,
  "os_disk": {
    "name": "object-name",
    "operating_system_type": "Linux"
  },
  "plan": null,
  "powerstate": "running",
  "private_ip": "172.26.3.6",
  "private_ip_alloc_method": "Static",
  "provisioning_state": "Succeeded",
  "public_ip": "XXX.XXX.XXX.XXX",
  "public_ip_alloc_method": "Static",
  "public_ip_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/
↪providers/Microsoft.Network/publicIPAddresses/object-name",
  "public_ip_name": "object-name",
  "resource_group": "galaxy-production",
  "security_group": "object-name",
  "security_group_id": "/subscriptions/subscription-id/resourceGroups/galaxy-production/
↪providers/Microsoft.Network/networkSecurityGroups/object-name",
  "tags": {
    "db": "mysql"
  },
  "type": "Microsoft.Compute/virtualMachines",
  "virtual_machine_size": "Standard_DS4"
}
```

Host Groups

By default hosts are grouped by:

- azure (all hosts)
- location name
- resource group name
- security group name
- tag key
- tag key_value
- os_disk operating_system_type (Windows/Linux)

You can control host groupings and host selection by either defining environment variables or creating an azure_rm.ini file in your current working directory.

NOTE: An .ini file will take precedence over environment variables.

NOTE: The name of the .ini file is the basename of the inventory script (in other words, 'azure_rm') with a '.ini' extension. This allows you to copy, rename and customize the inventory script and have matching .ini files all in the same directory.

Control grouping using the following variables defined in the environment:

- AZURE_GROUP_BY_RESOURCE_GROUP=yes
- AZURE_GROUP_BY_LOCATION=yes
- AZURE_GROUP_BY_SECURITY_GROUP=yes
- AZURE_GROUP_BY_TAG=yes
- AZURE_GROUP_BY_OS_FAMILY=yes

Select hosts within specific resource groups by assigning a comma separated list to:

- AZURE_RESOURCE_GROUPS=resource_group_a,resource_group_b

Select hosts for specific tag key by assigning a comma separated list of tag keys to:

- AZURE_TAGS=key1,key2,key3

Select hosts for specific locations by assigning a comma separated list of locations to:

- AZURE_LOCATIONS=eastus,eastus2,westus

Or, select hosts for specific tag key:value pairs by assigning a comma separated list key:value pairs to:

- AZURE_TAGS=key1:value1,key2:value2

If you don't need the powerstate, you can improve performance by turning off powerstate fetching:

- AZURE_INCLUDE_POWERSTATE=no

A sample azure_rm.ini file is included along with the inventory script in here. An .ini file will contain the following:

```
[azure]
# Control which resource groups are included. By default all resources groups are␣
↪included.
# Set resource_groups to a comma separated list of resource groups names.
#resource_groups=

# Control which tags are included. Set tags to a comma separated list of keys or␣
↪key:value pairs
#tags=

# Control which locations are included. Set locations to a comma separated list of␣
↪locations.
#locations=

# Include powerstate. If you don't need powerstate information, turning it off improves␣
↪runtime performance.
# Valid values: yes, no, true, false, True, False, 0, 1.
include_powerstate=yes

# Control grouping with the following boolean flags. Valid values: yes, no, true, false,␣
↪True, False, 0, 1.
group_by_resource_group=yes
group_by_location=yes
group_by_security_group=yes
```

(continues on next page)

(continued from previous page)

```
group_by_tag=yes
group_by_os_family=yes
```

Examples

Here are some examples using the inventory script:

```
# Download inventory script
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/
↪inventory/azure_rm.py

# Execute /bin/uname on all instances in the Testing resource group
$ ansible -i azure_rm.py Testing -m shell -a "/bin/uname -a"

# Execute win_ping on all Windows instances
$ ansible -i azure_rm.py windows -m win_ping

# Execute ping on all Linux instances
$ ansible -i azure_rm.py linux -m ping

# Use the inventory script to print instance specific information
$ ./azure_rm.py --host my_instance_host_name --resource-groups=Testing --pretty

# Use the inventory script with ansible-playbook
$ ansible-playbook -i ./azure_rm.py test_playbook.yml
```

Here is a simple playbook to exercise the Azure inventory script:

```
- name: Test the inventory script
  hosts: azure
  connection: local
  gather_facts: false
  tasks:
    - debug:
        msg: "{{ inventory_hostname }} has powerstate {{ powerstate }}"
```

You can execute the playbook with something like:

```
$ ansible-playbook -i ./azure_rm.py test_azure_inventory.yml
```

Disabling certificate validation on Azure endpoints

When an HTTPS proxy is present, or when using Azure Stack, it may be necessary to disable certificate validation for Azure endpoints in the Azure modules. This is not a recommended security practice, but may be necessary when the system CA store cannot be altered to include the necessary CA certificate. Certificate validation can be controlled by setting the “cert_validation_mode” value in a credential profile, through the “AZURE_CERT_VALIDATION_MODE” environment variable, or by passing the “cert_validation_mode” argument to any Azure module. The default value is “validate”; setting the value to “ignore” will prevent all certificate validation. The module argument takes precedence over a credential profile value, which takes precedence over the environment value.

1.18.5 Online.net Guide

Introduction

Online is a French hosting company mainly known for providing bare-metal servers named Dedibox. Check it out: https://www.online.net/en

Dynamic inventory for Online resources

Ansible has a dynamic inventory plugin that can list your resources.

1. Create a YAML configuration such as `online_inventory.yml` with this content:

```
plugin: online
```

2. **Set your `ONLINE_TOKEN` environment variable with your token.**
   You need to open an account and log into it before you can get a token. You can find your token at the following page: https://console.online.net/en/api/access

3. You can test that your inventory is working by running:

```
$ ansible-inventory -v -i online_inventory.yml --list
```

4. Now you can run your playbook or any other module with this inventory:

```
$ ansible all -i online_inventory.yml -m ping
sd-96735 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

1.18.6 Oracle Cloud Infrastructure Guide

Introduction

Oracle provides a number of Ansible modules to interact with Oracle Cloud Infrastructure (OCI). In this guide, we will explain how you can use these modules to orchestrate, provision and configure your infrastructure on OCI.

Requirements

To use the OCI Ansible modules, you must have the following prerequisites on your control node, the computer from which Ansible playbooks are executed.

1. An Oracle Cloud Infrastructure account.
2. A user created in that account, in a security group with a policy that grants the necessary permissions for working with resources in those compartments. For guidance, see How Policies Work.
3. The necessary credentials and OCID information.

Installation

1. Install the Oracle Cloud Infrastructure Python SDK (detailed installation instructions):

```
pip install oci
```

2. Install the Ansible OCI Modules in one of two ways:

a. From Galaxy:

```
ansible-galaxy install oracle.oci_ansible_modules
```

b. From GitHub:

```
$ git clone https://github.com/oracle/oci-ansible-modules.git
```

```
$ cd oci-ansible-modules
```

Run one of the following commands:

- If Ansible is installed only for your user:

```
$ ./install.py
```

- If Ansible is installed as root:

```
$ sudo ./install.py
```

Configuration

When creating and configuring Oracle Cloud Infrastructure resources, Ansible modules use the authentication information outlined here. .

Examples

Launch a compute instance

This sample launch playbook launches a public Compute instance and then accesses the instance from an Ansible module over an SSH connection. The sample illustrates how to:

- Generate a temporary, host-specific SSH key pair.
- Specify the public key from the key pair for connecting to the instance, and then launch the instance.
- Connect to the newly launched instance using SSH.

Create and manage Autonomous Data Warehouses

This sample warehouse playbook creates an Autonomous Data Warehouse and manage its lifecycle. The sample shows how to:

- Set up an Autonomous Data Warehouse.
- List all of the Autonomous Data Warehouse instances available in a compartment, filtered by the display name.
- Get the "facts" for a specified Autonomous Data Warehouse.
- Stop and start an Autonomous Data Warehouse instance.
- Delete an Autonomous Data Warehouse instance.

Create and manage Autonomous Transaction Processing

This sample playbook creates an Autonomous Transaction Processing database and manage its lifecycle. The sample shows how to:

- Set up an Autonomous Transaction Processing database instance.
- List all of the Autonomous Transaction Processing instances in a compartment, filtered by the display name.
- Get the "facts" for a specified Autonomous Transaction Processing instance.
- Delete an Autonomous Transaction Processing database instance.

You can find more examples here: Sample Ansible Playbooks.

1.18.7 Packet.net Guide

Introduction

Packet.net is a bare metal infrastructure host that's supported by Ansible (>=2.3) through a dynamic inventory script and two cloud modules. The two modules are:

- packet_sshkey: adds a public SSH key from file or value to the Packet infrastructure. Every subsequently-created device will have this public key installed in .ssh/authorized_keys.
- packet_device: manages servers on Packet. You can use this module to create, restart and delete devices.

Note, this guide assumes you are familiar with Ansible and how it works. If you're not, have a look at their *docs* before getting started.

Requirements

The Packet modules and inventory script connect to the Packet API using the packet-python package. You can install it with pip:

```
$ pip install packet-python
```

In order to check the state of devices created by Ansible on Packet, it's a good idea to install one of the Packet CLI clients. Otherwise you can check them through the Packet portal.

To use the modules and inventory script you'll need a Packet API token. You can generate an API token through the Packet portal here. The simplest way to authenticate yourself is to set the Packet API token in an environment variable:

```
$ export PACKET_API_TOKEN=Bfse9F24SFtfs423Gsd3ifGsd43sSdfs
```

If you're not comfortable exporting your API token, you can pass it as a parameter to the modules.

On Packet, devices and reserved IP addresses belong to projects. In order to use the packet_device module, you need to specify the UUID of the project in which you want to create or manage devices. You can find a project's UUID in the Packet portal here (it's just under the project table) or through one of the available CLIs.

If you want to use a new SSH key pair in this tutorial, you can generate it to `./id_rsa` and `./id_rsa.pub` as:

```
$ ssh-keygen -t rsa -f ./id_rsa
```

If you want to use an existing key pair, just copy the private and public key over to the playbook directory.

Device Creation

The following code block is a simple playbook that creates one Type 0 server (the 'plan' parameter). You have to supply 'plan' and 'operating_system'. 'location' defaults to 'ewr1' (Parsippany, NJ). You can find all the possible values for the parameters through a CLI client.

```
# playbook_create.yml

- name: create ubuntu device
  hosts: localhost
  tasks:

  - packet_sshkey:
      key_file: ./id_rsa.pub
      label: tutorial key

  - packet_device:
      project_id: <your_project_id>
      hostnames: myserver
      operating_system: ubuntu_16_04
      plan: baremetal_0
      facility: sjc1
```

After running `ansible-playbook playbook_create.yml`, you should have a server provisioned on Packet. You can verify through a CLI or in the Packet portal.

If you get an error with the message "failed to set machine state present, error: Error 404: Not Found", please verify your project UUID.

Updating Devices

The two parameters used to uniquely identify Packet devices are: "device_ids" and "hostnames". Both parameters accept either a single string (later converted to a one-element list), or a list of strings.

The 'device_ids' and 'hostnames' parameters are mutually exclusive. The following values are all acceptable:

- device_ids: a27b7a83-fc93-435b-a128-47a5b04f2dcf
- hostnames: mydev1
- device_ids: [a27b7a83-fc93-435b-a128-47a5b04f2dcf, 4887130f-0ccd-49a0-99b0-323c1ceb527b]

- hostnames: [mydev1, mydev2]

In addition, hostnames can contain a special '%d' formatter along with a 'count' parameter that lets you easily expand hostnames that follow a simple name and number pattern; in other words, `hostnames: "mydev%d", count: 2` will expand to [mydev1, mydev2].

If your playbook acts on existing Packet devices, you can only pass the 'hostname' and 'device_ids' parameters. The following playbook shows how you can reboot a specific Packet device by setting the 'hostname' parameter:

```
# playbook_reboot.yml

- name: reboot myserver
  hosts: localhost
  tasks:

  - packet_device:
      project_id: <your_project_id>
      hostnames: myserver
      state: rebooted
```

You can also identify specific Packet devices with the 'device_ids' parameter. The device's UUID can be found in the Packet Portal or by using a CLI. The following playbook removes a Packet device using the 'device_ids' field:

```
# playbook_remove.yml

- name: remove a device
  hosts: localhost
  tasks:

  - packet_device:
      project_id: <your_project_id>
      device_ids: <myserver_device_id>
      state: absent
```

More Complex Playbooks

In this example, we'll create a CoreOS cluster with user data.

The CoreOS cluster will use etcd for discovery of other servers in the cluster. Before provisioning your servers, you'll need to generate a discovery token for your cluster:

```
$ curl -w "\n" 'https://discovery.etcd.io/new?size=3'
```

The following playbook will create an SSH key, 3 Packet servers, and then wait until SSH is ready (or until 5 minutes passed). Make sure to substitute the discovery token URL in 'user_data', and the 'project_id' before running `ansible-playbook`. Also, feel free to change 'plan' and 'facility'.

```
# playbook_coreos.yml

- name: Start 3 CoreOS nodes in Packet and wait until SSH is ready
  hosts: localhost
  tasks:

  - packet_sshkey:
```

(continues on next page)

(continued from previous page)

```
      key_file: ./id_rsa.pub
      label: new

  - packet_device:
      hostnames: [coreos-one, coreos-two, coreos-three]
      operating_system: coreos_beta
      plan: baremetal_0
      facility: ewr1
      project_id: <your_project_id>
      wait_for_public_IPv: 4
      user_data: |
        #cloud-config
        coreos:
          etcd2:
            discovery: https://discovery.etcd.io/<token>
            advertise-client-urls: http://$private_ipv4:2379,http://$private_ipv4:4001
            initial-advertise-peer-urls: http://$private_ipv4:2380
            listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
            listen-peer-urls: http://$private_ipv4:2380
          fleet:
            public-ip: $private_ipv4
          units:
            - name: etcd2.service
              command: start
            - name: fleet.service
              command: start
    register: newhosts

  - name: wait for ssh
    wait_for:
      delay: 1
      host: "{{ item.public_ipv4 }}"
      port: 22
      state: started
      timeout: 500
    loop: "{{ newhosts.results[0].devices }}"
```

As with most Ansible modules, the default states of the Packet modules are idempotent, meaning the resources in your project will remain the same after re-runs of a playbook. Thus, we can keep the `packet_sshkey` module call in our playbook. If the public key is already in your Packet account, the call will have no effect.

The second module call provisions 3 Packet Type 0 (specified using the 'plan' parameter) servers in the project identified by the 'project_id' parameter. The servers are all provisioned with CoreOS beta (the 'operating_system' parameter) and are customized with cloud-config user data passed to the 'user_data' parameter.

The `packet_device` module has a `wait_for_public_IPv` that is used to specify the version of the IP address to wait for (valid values are `4` or `6` for IPv4 or IPv6). If specified, Ansible will wait until the GET API call for a device contains an Internet-routeable IP address of the specified version. When referring to an IP address of a created device in subsequent module calls, it's wise to use the `wait_for_public_IPv` parameter, or `state: active` in the packet_device module call.

Run the playbook:

```
$ ansible-playbook playbook_coreos.yml
```

Once the playbook quits, your new devices should be reachable through SSH. Try to connect to one and check if etcd has started properly:

```
tomk@work $ ssh -i id_rsa core@$one_of_the_servers_ip
core@coreos-one ~ $ etcdctl cluster-health
```

Once you create a couple of devices, you might appreciate the dynamic inventory script…

Dynamic Inventory Script

The dynamic inventory script queries the Packet API for a list of hosts, and exposes it to Ansible so you can easily identify and act on Packet devices.

You can find it in Ansible Community General Collection's git repo at scripts/inventory/packet_net.py.

The inventory script is configurable through an ini file.

If you want to use the inventory script, you must first export your Packet API token to a PACKET_API_TOKEN environment variable.

You can either copy the inventory and ini config out from the cloned git repo, or you can download it to your working directory like so:

```
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/
↪inventory/packet_net.py
$ chmod +x packet_net.py
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/
↪inventory/packet_net.ini
```

In order to understand what the inventory script gives to Ansible you can run:

```
$ ./packet_net.py --list
```

It should print a JSON document looking similar to following trimmed dictionary:

```
{
  "_meta": {
    "hostvars": {
      "147.75.64.169": {
        "packet_billing_cycle": "hourly",
        "packet_created_at": "2017-02-09T17:11:26Z",
        "packet_facility": "ewr1",
        "packet_hostname": "coreos-two",
        "packet_href": "/devices/d0ab8972-54a8-4bff-832b-28549d1bec96",
        "packet_id": "d0ab8972-54a8-4bff-832b-28549d1bec96",
        "packet_locked": false,
        "packet_operating_system": "coreos_beta",
        "packet_plan": "baremetal_0",
        "packet_state": "active",
        "packet_updated_at": "2017-02-09T17:16:35Z",
        "packet_user": "core",
        "packet_userdata": "#cloud-config\ncoreos:\n  etcd2:\n    discovery: https://
↪discovery.etcd.io/e0c8a4a9b8fe61acd51ec599e2a4f68e\n    advertise-client-urls: http://
```

(continues on next page)

(continued from previous page)

```
↪$private_ipv4:2379,http://$private_ipv4:4001\n    initial-advertise-peer-urls: http://
↪$private_ipv4:2380\n    listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001\n ␣
↪  listen-peer-urls: http://$private_ipv4:2380\n  fleet:\n    public-ip: $private_ipv4\
↪n  units:\n    - name: etcd2.service\n      command: start\n    - name: fleet.service\
↪n      command: start"
      }
    }
  },
  "baremetal_0": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249",
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "coreos_beta": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249",
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "ewr1": [
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "sjc1": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249"
  ],
  "coreos-two": [
    "147.75.64.169"
  ],
  "d0ab8972-54a8-4bff-832b-28549d1bec96": [
    "147.75.64.169"
  ]
}
```

In the `['_meta']['hostvars']` key, there is a list of devices (uniquely identified by their public IPv4 address) with their parameters. The other keys under `['_meta']` are lists of devices grouped by some parameter. Here, it is type (all devices are of type baremetal_0), operating system, and facility (ewr1 and sjc1).

In addition to the parameter groups, there are also one-item groups with the UUID or hostname of the device.

You can now target groups in playbooks! The following playbook will install a role that supplies resources for an Ansible target into all devices in the "coreos_beta" group:

```
# playbook_bootstrap.yml
```

(continues on next page)

(continued from previous page)

```
- hosts: coreos_beta
  gather_facts: false
  roles:
    - defunctzombie.coreos-boostrap
```

Don't forget to supply the dynamic inventory in the `-i` argument!

```
$ ansible-playbook -u core -i packet_net.py playbook_bootstrap.yml
```

If you have any questions or comments let us know! help@packet.net

1.18.8 Rackspace Cloud Guide

Introduction

Note: Rackspace functionality in Ansible is not maintained and users should consider the OpenStack collection instead.

Ansible contains a number of core modules for interacting with Rackspace Cloud.

The purpose of this section is to explain how to put Ansible modules together (and use inventory scripts) to use Ansible in a Rackspace Cloud context.

Prerequisites for using the rax modules are minimal. In addition to ansible itself, all of the modules require and are tested against pyrax 1.5 or higher. You'll need this Python module installed on the execution host.

`pyrax` is not currently available in many operating system package repositories, so you will likely need to install it through pip:

```
$ pip install pyrax
```

Ansible creates an implicit localhost that executes in the same context as the `ansible-playbook` and the other CLI tools. If for any reason you need or want to have it in your inventory you should do something like the following:

```
[localhost]
localhost ansible_connection=local ansible_python_interpreter=/usr/local/bin/python2
```

For more information see Implicit Localhost

In playbook steps, we'll typically be using the following pattern:

```
- hosts: localhost
  gather_facts: False
  tasks:
```

Credentials File

The *rax.py* inventory script and all *rax* modules support a standard *pyrax* credentials file that looks like:

```
[rackspace_cloud]
username = myraxusername
api_key = d41d8cd98f00b204e9800998ecf8427e
```

Setting the environment parameter `RAX_CREDS_FILE` to the path of this file will help Ansible find how to load this information.

More information about this credentials file can be found at https://github.com/pycontribs/pyrax/blob/master/docs/getting_started.md#authenticating

Running from a Python Virtual Environment (Optional)

Most users will not be using virtualenv, but some users, particularly Python developers sometimes like to.

There are special considerations when Ansible is installed to a Python virtualenv, rather than the default of installing at a global scope. Ansible assumes, unless otherwise instructed, that the python binary will live at /usr/bin/python. This is done through the interpreter line in modules, however when instructed by setting the inventory variable 'ansible_python_interpreter', Ansible will use this specified path instead to find Python. This can be a cause of confusion as one may assume that modules running on 'localhost', or perhaps running through 'local_action', are using the virtualenv Python interpreter. By setting this line in the inventory, the modules will execute in the virtualenv interpreter and have available the virtualenv packages, specifically pyrax. If using virtualenv, you may wish to modify your localhost inventory definition to find this location as follows:

```
[localhost]
localhost ansible_connection=local ansible_python_interpreter=/path/to/ansible_venv/bin/
↪python
```

Note: pyrax may be installed in the global Python package scope or in a virtual environment. There are no special considerations to keep in mind when installing pyrax.

Provisioning

Now for the fun parts.

The 'rax' module provides the ability to provision instances within Rackspace Cloud. Typically the provisioning task will be performed from your Ansible control server (in our example, localhost) against the Rackspace cloud API. This is done for several reasons:

- Avoiding installing the pyrax library on remote nodes
- No need to encrypt and distribute credentials to remote nodes
- Speed and simplicity

Note: Authentication with the Rackspace-related modules is handled by either specifying your username and API key as environment variables or passing them as module arguments, or by specifying the location of a credentials file.

Here is a basic example of provisioning an instance in ad hoc mode:

```
$ ansible localhost -m rax -a "name=awx flavor=4 image=ubuntu-1204-lts-precise-pangolin␣
↪wait=yes"
```

Here's what it would look like in a playbook, assuming the parameters were defined in variables:

```
tasks:
  - name: Provision a set of instances
    rax:
        name: "{{ rax_name }}"
        flavor: "{{ rax_flavor }}"
        image: "{{ rax_image }}"
        count: "{{ rax_count }}"
        group: "{{ group }}"
        wait: true
    register: rax
    delegate_to: localhost
```

The rax module returns data about the nodes it creates, like IP addresses, hostnames, and login passwords. By registering the return value of the step, it is possible used this data to dynamically add the resulting hosts to inventory (temporarily, in memory). This facilitates performing configuration actions on the hosts in a follow-on task. In the following example, the servers that were successfully created using the above task are dynamically added to a group called "raxhosts", with each nodes hostname, IP address, and root password being added to the inventory.

```
- name: Add the instances we created (by public IP) to the group 'raxhosts'
  add_host:
      hostname: "{{ item.name }}"
      ansible_host: "{{ item.rax_accessipv4 }}"
      ansible_password: "{{ item.rax_adminpass }}"
      groups: raxhosts
  loop: "{{ rax.success }}"
  when: rax.action == 'create'
```

With the host group now created, the next play in this playbook could now configure servers belonging to the raxhosts group.

```
- name: Configuration play
  hosts: raxhosts
  user: root
  roles:
    - ntp
    - webserver
```

The method above ties the configuration of a host with the provisioning step. This isn't always what you want, and leads us to the next section.

Host Inventory

Once your nodes are spun up, you'll probably want to talk to them again. The best way to handle this is to use the "rax" inventory plugin, which dynamically queries Rackspace Cloud and tells Ansible what nodes you have to manage. You might want to use this even if you are spinning up cloud instances through other tools, including the Rackspace Cloud user interface. The inventory plugin can be used to group resources by metadata, region, OS, and so on. Utilizing metadata is highly recommended in "rax" and can provide an easy way to sort between host groups and roles. If you don't want to use the `rax.py` dynamic inventory script, you could also still choose to manually manage your INI inventory file, though this is less recommended.

In Ansible it is quite possible to use multiple dynamic inventory plugins along with INI file data. Just put them in a common directory and be sure the scripts are chmod +x, and the INI-based ones are not.

rax.py

To use the Rackspace dynamic inventory script, copy `rax.py` into your inventory directory and make it executable. You can specify a credentials file for `rax.py` utilizing the `RAX_CREDS_FILE` environment variable.

Note: Dynamic inventory scripts (like `rax.py`) are saved in `/usr/share/ansible/inventory` if Ansible has been installed globally. If installed to a virtualenv, the inventory scripts are installed to `$VIRTUALENV/share/inventory`.

Note: Users of *Red Hat Ansible Automation Platform* will note that dynamic inventory is natively supported by the controller in the platform, and all you have to do is associate a group with your Rackspace Cloud credentials, and it will easily synchronize without going through these steps:

```
$ RAX_CREDS_FILE=~/.raxpub ansible all -i rax.py -m setup
```

`rax.py` also accepts a `RAX_REGION` environment variable, which can contain an individual region, or a comma separated list of regions.

When using `rax.py`, you will not have a 'localhost' defined in the inventory.

As mentioned previously, you will often be running most of these modules outside of the host loop, and will need 'localhost' defined. The recommended way to do this, would be to create an `inventory` directory, and place both the `rax.py` script and a file containing `localhost` in it.

Executing `ansible` or `ansible-playbook` and specifying the `inventory` directory instead of an individual file, will cause ansible to evaluate each file in that directory for inventory.

Let's test our inventory script to see if it can talk to Rackspace Cloud.

```
$ RAX_CREDS_FILE=~/.raxpub ansible all -i inventory/ -m setup
```

Assuming things are properly configured, the `rax.py` inventory script will output information similar to the following information, which will be utilized for inventory and variables.

```
{
    "ORD": [
        "test"
    ],
    "_meta": {
        "hostvars": {
```

(continues on next page)

(continued from previous page)

```
            "test": {
                "ansible_host": "198.51.100.1",
                "rax_accessipv4": "198.51.100.1",
                "rax_accessipv6": "2001:DB8::2342",
                "rax_addresses": {
                    "private": [
                        {
                            "addr": "192.0.2.2",
                            "version": 4
                        }
                    ],
                    "public": [
                        {
                            "addr": "198.51.100.1",
                            "version": 4
                        },
                        {
                            "addr": "2001:DB8::2342",
                            "version": 6
                        }
                    ]
                },
                "rax_config_drive": "",
                "rax_created": "2013-11-14T20:48:22Z",
                "rax_flavor": {
                    "id": "performance1-1",
                    "links": [
                        {
                            "href": "https://ord.servers.api.rackspacecloud.com/111111/
↪flavors/performance1-1",
                            "rel": "bookmark"
                        }
                    ]
                },
                "rax_hostid":
↪"e7b6961a9bd943ee82b13816426f1563bfda6846aad84d52af45a4904660cde0",
                "rax_human_id": "test",
                "rax_id": "099a447b-a644-471f-87b9-a7f580eb0c2a",
                "rax_image": {
                    "id": "b211c7bf-b5b4-4ede-a8de-a4368750c653",
                    "links": [
                        {
                            "href": "https://ord.servers.api.rackspacecloud.com/111111/
↪images/b211c7bf-b5b4-4ede-a8de-a4368750c653",
                            "rel": "bookmark"
                        }
                    ]
                },
                "rax_key_name": null,
                "rax_links": [
                    {
                        "href": "https://ord.servers.api.rackspacecloud.com/v2/111111/
```

(continues on next page)

(continued from previous page)

```
↪servers/099a447b-a644-471f-87b9-a7f580eb0c2a",
                                "rel": "self"
                            },
                            {
                                "href": "https://ord.servers.api.rackspacecloud.com/111111/
↪servers/099a447b-a644-471f-87b9-a7f580eb0c2a",
                                "rel": "bookmark"
                            }
                        ],
                        "rax_metadata": {
                            "foo": "bar"
                        },
                        "rax_name": "test",
                        "rax_name_attr": "name",
                        "rax_networks": {
                            "private": [
                                "192.0.2.2"
                            ],
                            "public": [
                                "198.51.100.1",
                                "2001:DB8::2342"
                            ]
                        },
                        "rax_os-dcf_diskconfig": "AUTO",
                        "rax_os-ext-sts_power_state": 1,
                        "rax_os-ext-sts_task_state": null,
                        "rax_os-ext-sts_vm_state": "active",
                        "rax_progress": 100,
                        "rax_status": "ACTIVE",
                        "rax_tenant_id": "111111",
                        "rax_updated": "2013-11-14T20:49:27Z",
                        "rax_user_id": "22222"
                    }
                }
            }
        }
```

Standard Inventory

When utilizing a standard ini formatted inventory file (as opposed to the inventory plugin), it may still be advantageous to retrieve discoverable hostvar information from the Rackspace API.

This can be achieved with the `rax_facts` module and an inventory file similar to the following:

```
[test_servers]
hostname1 rax_region=ORD
hostname2 rax_region=ORD
```

```
- name: Gather info about servers
  hosts: test_servers
  gather_facts: False
```

(continues on next page)

(continued from previous page)

```
  tasks:
    - name: Get facts about servers
      rax_facts:
        credentials: ~/.raxpub
        name: "{{ inventory_hostname }}"
        region: "{{ rax_region }}"
      delegate_to: localhost
    - name: Map some facts
      set_fact:
        ansible_host: "{{ rax_accessipv4 }}"
```

While you don't need to know how it works, it may be interesting to know what kind of variables are returned.

The `rax_facts` module provides facts as following, which match the `rax.py` inventory script:

```
{
    "ansible_facts": {
        "rax_accessipv4": "198.51.100.1",
        "rax_accessipv6": "2001:DB8::2342",
        "rax_addresses": {
            "private": [
                {
                    "addr": "192.0.2.2",
                    "version": 4
                }
            ],
            "public": [
                {
                    "addr": "198.51.100.1",
                    "version": 4
                },
                {
                    "addr": "2001:DB8::2342",
                    "version": 6
                }
            ]
        },
        "rax_config_drive": "",
        "rax_created": "2013-11-14T20:48:22Z",
        "rax_flavor": {
            "id": "performance1-1",
            "links": [
                {
                    "href": "https://ord.servers.api.rackspacecloud.com/111111/flavors/
↪performance1-1",
                    "rel": "bookmark"
                }
            ]
        },
        "rax_hostid": "e7b6961a9bd943ee82b13816426f1563bfda6846aad84d52af45a4904660cde0",
        "rax_human_id": "test",
        "rax_id": "099a447b-a644-471f-87b9-a7f580eb0c2a",
        "rax_image": {
```

(continues on next page)

(continued from previous page)

```
            "id": "b211c7bf-b5b4-4ede-a8de-a4368750c653",
            "links": [
                {
                    "href": "https://ord.servers.api.rackspacecloud.com/111111/images/
↪b211c7bf-b5b4-4ede-a8de-a4368750c653",
                    "rel": "bookmark"
                }
            ]
        },
        "rax_key_name": null,
        "rax_links": [
            {
                "href": "https://ord.servers.api.rackspacecloud.com/v2/111111/servers/
↪099a447b-a644-471f-87b9-a7f580eb0c2a",
                "rel": "self"
            },
            {
                "href": "https://ord.servers.api.rackspacecloud.com/111111/servers/
↪099a447b-a644-471f-87b9-a7f580eb0c2a",
                "rel": "bookmark"
            }
        ],
        "rax_metadata": {
            "foo": "bar"
        },
        "rax_name": "test",
        "rax_name_attr": "name",
        "rax_networks": {
            "private": [
                "192.0.2.2"
            ],
            "public": [
                "198.51.100.1",
                "2001:DB8::2342"
            ]
        },
        "rax_os-dcf_diskconfig": "AUTO",
        "rax_os-ext-sts_power_state": 1,
        "rax_os-ext-sts_task_state": null,
        "rax_os-ext-sts_vm_state": "active",
        "rax_progress": 100,
        "rax_status": "ACTIVE",
        "rax_tenant_id": "111111",
        "rax_updated": "2013-11-14T20:49:27Z",
        "rax_user_id": "22222"
    },
    "changed": false
}
```

Use Cases

This section covers some additional usage examples built around a specific use case.

Network and Server

Create an isolated cloud network and build a server

```
- name: Build Servers on an Isolated Network
  hosts: localhost
  gather_facts: False
  tasks:
    - name: Network create request
      rax_network:
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        region: IAD
        state: present
      delegate_to: localhost

    - name: Server create request
      rax:
        credentials: ~/.raxpub
        name: web%04d.example.org
        flavor: 2
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        networks:
          - public
          - my-net
        region: IAD
        state: present
        count: 5
        exact_count: true
        group: web
        wait: true
        wait_timeout: 360
      register: rax
      delegate_to: localhost
```

Complete Environment

Build a complete webserver environment with servers, custom networks and load balancers, install nginx and create a custom index.html

```
---
- name: Build environment
  hosts: localhost
  gather_facts: False
  tasks:
```

(continues on next page)

(continued from previous page)

```
    - name: Load Balancer create request
      rax_clb:
        credentials: ~/.raxpub
        name: my-lb
        port: 80
        protocol: HTTP
        algorithm: ROUND_ROBIN
        type: PUBLIC
        timeout: 30
        region: IAD
        wait: true
        state: present
        meta:
          app: my-cool-app
      register: clb

    - name: Network create request
      rax_network:
        credentials: ~/.raxpub
        label: my-net
        cidr: 192.168.3.0/24
        state: present
        region: IAD
      register: network

    - name: Server create request
      rax:
        credentials: ~/.raxpub
        name: web%04d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        networks:
          - public
          - private
          - my-net
        region: IAD
        state: present
        count: 5
        exact_count: true
        group: web
        wait: true
      register: rax

    - name: Add servers to web host group
      add_host:
        hostname: "{{ item.name }}"
        ansible_host: "{{ item.rax_accessipv4 }}"
        ansible_password: "{{ item.rax_adminpass }}"
        ansible_user: root
        groups: web
      loop: "{{ rax.success }}"
```

(continues on next page)

(continued from previous page)

```
      when: rax.action == 'create'

    - name: Add servers to Load balancer
      rax_clb_nodes:
        credentials: ~/.raxpub
        load_balancer_id: "{{ clb.balancer.id }}"
        address: "{{ item.rax_networks.private|first }}"
        port: 80
        condition: enabled
        type: primary
        wait: true
        region: IAD
      loop: "{{ rax.success }}"
      when: rax.action == 'create'

- name: Configure servers
  hosts: web
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted

  tasks:
    - name: Install nginx
      apt: pkg=nginx state=latest update_cache=yes cache_valid_time=86400
      notify:
        - restart nginx

    - name: Ensure nginx starts on boot
      service: name=nginx state=started enabled=yes

    - name: Create custom index.html
      copy: content="{{ inventory_hostname }}" dest=/usr/share/nginx/www/index.html
            owner=root group=root mode=0644
```

RackConnect and Managed Cloud

When using RackConnect version 2 or Rackspace Managed Cloud there are Rackspace automation tasks that are executed on the servers you create after they are successfully built. If your automation executes before the RackConnect or Managed Cloud automation, you can cause failures and unusable servers.

These examples show creating servers, and ensuring that the Rackspace automation has completed before Ansible continues onwards.

For simplicity, these examples are joined, however both are only needed when using RackConnect. When only using Managed Cloud, the RackConnect portion can be ignored.

The RackConnect portions only apply to RackConnect version 2.

Using a Control Machine

```
- name: Create an exact count of servers
  hosts: localhost
  gather_facts: False
  tasks:
    - name: Server build requests
      rax:
        credentials: ~/.raxpub
        name: web%03d.example.org
        flavor: performance1-1
        image: ubuntu-1204-lts-precise-pangolin
        disk_config: manual
        region: DFW
        state: present
        count: 1
        exact_count: true
        group: web
        wait: true
      register: rax

    - name: Add servers to in memory groups
      add_host:
        hostname: "{{ item.name }}"
        ansible_host: "{{ item.rax_accessipv4 }}"
        ansible_password: "{{ item.rax_adminpass }}"
        ansible_user: root
        rax_id: "{{ item.rax_id }}"
        groups: web,new_web
      loop: "{{ rax.success }}"
      when: rax.action == 'create'

- name: Wait for rackconnect and managed cloud automation to complete
  hosts: new_web
  gather_facts: false
  tasks:
    - name: ensure we run all tasks from localhost
      delegate_to: localhost
      block:
        - name: Wait for rackconnnect automation to complete
          rax_facts:
            credentials: ~/.raxpub
            id: "{{ rax_id }}"
            region: DFW
          register: rax_facts
          until: rax_facts.ansible_facts['rax_metadata']['rackconnect_automation_status
↪']|default('') == 'DEPLOYED'
          retries: 30
          delay: 10

        - name: Wait for managed cloud automation to complete
          rax_facts:
            credentials: ~/.raxpub
```

(continues on next page)

(continued from previous page)

```
          id: "{{ rax_id }}"
          region: DFW
        register: rax_facts
        until: rax_facts.ansible_facts['rax_metadata']['rax_service_level_automation
↪']|default('') == 'Complete'
        retries: 30
        delay: 10

- name: Update new_web hosts with IP that RackConnect assigns
  hosts: new_web
  gather_facts: false
  tasks:
    - name: Get facts about servers
      rax_facts:
        name: "{{ inventory_hostname }}"
        region: DFW
      delegate_to: localhost
    - name: Map some facts
      set_fact:
        ansible_host: "{{ rax_accessipv4 }}"

- name: Base Configure Servers
  hosts: web
  roles:
    - role: users

    - role: openssh
      opensshd_PermitRootLogin: "no"

    - role: ntp
```

Using Ansible Pull

```
---
- name: Ensure Rackconnect and Managed Cloud Automation is complete
  hosts: all
  tasks:
    - name: ensure we run all tasks from localhost
      delegate_to: localhost
      block:
        - name: Check for completed bootstrap
          stat:
            path: /etc/bootstrap_complete
          register: bootstrap

        - name: Get region
          command: xenstore-read vm-data/provider_data/region
          register: rax_region
          when: bootstrap.stat.exists != True
```

(continues on next page)

(continued from previous page)

```
        - name: Wait for rackconnect automation to complete
          uri:
            url: "https://{{ rax_region.stdout|trim }}.api.rackconnect.rackspace.com/v1/
↪automation_status?format=json"
            return_content: true
          register: automation_status
          when: bootstrap.stat.exists != True
          until: automation_status['automation_status']|default('') == 'DEPLOYED'
          retries: 30
          delay: 10

        - name: Wait for managed cloud automation to complete
          wait_for:
            path: /tmp/rs_managed_cloud_automation_complete
            delay: 10
          when: bootstrap.stat.exists != True

        - name: Set bootstrap completed
          file:
            path: /etc/bootstrap_complete
            state: touch
            owner: root
            group: root
            mode: 0400

- name: Base Configure Servers
  hosts: all
  roles:
    - role: users

    - role: openssh
      opensshd_PermitRootLogin: "no"

    - role: ntp
```

Using Ansible Pull with XenStore

```
---
- name: Ensure Rackconnect and Managed Cloud Automation is complete
  hosts: all
  tasks:
    - name: Check for completed bootstrap
      stat:
        path: /etc/bootstrap_complete
      register: bootstrap

    - name: Wait for rackconnect_automation_status xenstore key to exist
      command: xenstore-exists vm-data/user-metadata/rackconnect_automation_status
      register: rcas_exists
      when: bootstrap.stat.exists != True
```

(continues on next page)

(continued from previous page)

```
      failed_when: rcas_exists.rc|int > 1
      until: rcas_exists.rc|int == 0
      retries: 30
      delay: 10

    - name: Wait for rackconnect automation to complete
      command: xenstore-read vm-data/user-metadata/rackconnect_automation_status
      register: rcas
      when: bootstrap.stat.exists != True
      until: rcas.stdout|replace('"', '') == 'DEPLOYED'
      retries: 30
      delay: 10

    - name: Wait for rax_service_level_automation xenstore key to exist
      command: xenstore-exists vm-data/user-metadata/rax_service_level_automation
      register: rsla_exists
      when: bootstrap.stat.exists != True
      failed_when: rsla_exists.rc|int > 1
      until: rsla_exists.rc|int == 0
      retries: 30
      delay: 10

    - name: Wait for managed cloud automation to complete
      command: xenstore-read vm-data/user-metadata/rackconnect_automation_status
      register: rsla
      when: bootstrap.stat.exists != True
      until: rsla.stdout|replace('"', '') == 'DEPLOYED'
      retries: 30
      delay: 10

    - name: Set bootstrap completed
      file:
        path: /etc/bootstrap_complete
        state: touch
        owner: root
        group: root
        mode: 0400

- name: Base Configure Servers
  hosts: all
  roles:
    - role: users

    - role: openssh
      opensshd_PermitRootLogin: "no"

    - role: ntp
```

Advanced Usage

Autoscaling with AWX or Red Hat Ansible Automation Platform

The GUI component of Red Hat Ansible Automation Platform also contains a very nice feature for auto-scaling use cases. In this mode, a simple curl script can call a defined URL and the server will "dial out" to the requester and configure an instance that is spinning up. This can be a great way to reconfigure ephemeral nodes. See the documentation on provisioning callbacks for more details.

A benefit of using the callback approach over pull mode is that job results are still centrally recorded and less information has to be shared with remote hosts.

Orchestration in the Rackspace Cloud

Ansible is a powerful orchestration tool, and rax modules allow you the opportunity to orchestrate complex tasks, deployments, and configurations. The key here is to automate provisioning of infrastructure, like any other piece of software in an environment. Complex deployments might have previously required manual manipulation of load balancers, or manual provisioning of servers. Utilizing the rax modules included with Ansible, one can make the deployment of additional nodes contingent on the current number of running nodes, or the configuration of a clustered application dependent on the number of nodes with common metadata. One could automate the following scenarios, for example:

- Servers that are removed from a Cloud Load Balancer one-by-one, updated, verified, and returned to the load balancer pool
- Expansion of an already-online environment, where nodes are provisioned, bootstrapped, configured, and software installed
- A procedure where app log files are uploaded to a central location, like Cloud Files, before a node is decommissioned
- Servers and load balancers that have DNS records created and destroyed on creation and decommissioning, respectively

1.18.9 Scaleway Guide

Introduction

Scaleway is a cloud provider supported by Ansible, version 2.6 or higher through a dynamic inventory plugin and modules. Those modules are:

- scaleway_sshkey – Scaleway SSH keys management module: adds a public SSH key from a file or value to the Packet infrastructure. Every subsequently-created device will have this public key installed in .ssh/authorized_keys.
- scaleway_compute – Scaleway compute management module: manages servers on Scaleway. You can use this module to create, restart and delete servers.
- scaleway_volume – Scaleway volumes management module: manages volumes on Scaleway.

Note: This guide assumes you are familiar with Ansible and how it works. If you're not, have a look at ansible_documentation before getting started.

Requirements

The Scaleway modules and inventory script connect to the Scaleway API using Scaleway REST API. To use the modules and inventory script you'll need a Scaleway API token. You can generate an API token through the Scaleway console here. The simplest way to authenticate yourself is to set the Scaleway API token in an environment variable:

```
$ export SCW_TOKEN=00000000-1111-2222-3333-444444444444
```

If you're not comfortable exporting your API token, you can pass it as a parameter to the modules using the `api_token` argument.

If you want to use a new SSH key pair in this tutorial, you can generate it to `./id_rsa` and `./id_rsa.pub` as:

```
$ ssh-keygen -t rsa -f ./id_rsa
```

If you want to use an existing key pair, just copy the private and public key over to the playbook directory.

How to add an SSH key?

Connection to Scaleway Compute nodes use Secure Shell. SSH keys are stored at the account level, which means that you can re-use the same SSH key in multiple nodes. The first step to configure Scaleway compute resources is to have at least one SSH key configured.

scaleway_sshkey – Scaleway SSH keys management module is a module that manages SSH keys on your Scaleway account. You can add an SSH key to your account by including the following task in a playbook:

```
- name: "Add SSH key"
  scaleway_sshkey:
    ssh_pub_key: "ssh-rsa AAAA..."
    state: "present"
```

The `ssh_pub_key` parameter contains your ssh public key as a string. Here is an example inside a playbook:

```
- name: Test SSH key lifecycle on a Scaleway account
  hosts: localhost
  gather_facts: false
  environment:
    SCW_API_KEY: ""

  tasks:

    - scaleway_sshkey:
        ssh_pub_key: "ssh-rsa AAAAB...424242 developer@example.com"
        state: present
      register: result

    - assert:
        that:
          - result is success and result is changed
```

How to create a compute instance?

Now that we have an SSH key configured, the next step is to spin up a server! scaleway_compute – Scaleway compute management module is a module that can create, update and delete Scaleway compute instances:

```
- name: Create a server
  scaleway_compute:
    name: foobar
    state: present
    image: 00000000-1111-2222-3333-444444444444
    organization: 00000000-1111-2222-3333-444444444444
    region: ams1
    commercial_type: START1-S
```

Here are the parameter details for the example shown above:

- `name` is the name of the instance (the one that will show up in your web console).
- `image` is the UUID of the system image you would like to use. A list of all images is available for each availability zone.
- `organization` represents the organization that your account is attached to.
- `region` represents the Availability Zone which your instance is in (for this example, par1 and ams1).
- `commercial_type` represents the name of the commercial offers. You can check out the Scaleway pricing page to find which instance is right for you.

Take a look at this short playbook to see a working example using `scaleway_compute`:

```
- name: Test compute instance lifecycle on a Scaleway account
  hosts: localhost
  gather_facts: false
  environment:
    SCW_API_KEY: ""

  tasks:

    - name: Create a server
      register: server_creation_task
      scaleway_compute:
        name: foobar
        state: present
        image: 00000000-1111-2222-3333-444444444444
        organization: 00000000-1111-2222-3333-444444444444
        region: ams1
        commercial_type: START1-S
        wait: true

    - debug: var=server_creation_task

    - assert:
        that:
          - server_creation_task is success
          - server_creation_task is changed

    - name: Run it
```

(continues on next page)

(continued from previous page)

```
      scaleway_compute:
        name: foobar
        state: running
        image: 00000000-1111-2222-3333-444444444444
        organization: 00000000-1111-2222-3333-444444444444
        region: ams1
        commercial_type: START1-S
        wait: true
        tags:
          - web_server
      register: server_run_task

    - debug: var=server_run_task

    - assert:
        that:
          - server_run_task is success
          - server_run_task is changed
```

Dynamic Inventory Script

Ansible ships with scaleway – Scaleway inventory source. You can now get a complete inventory of your Scaleway resources through this plugin and filter it on different parameters (`regions` and `tags` are currently supported).

Let's create an example! Suppose that we want to get all hosts that got the tag web_server. Create a file named `scaleway_inventory.yml` with the following content:

```
plugin: scaleway
regions:
  - ams1
  - par1
tags:
  - web_server
```

This inventory means that we want all hosts that got the tag `web_server` on the zones `ams1` and `par1`. Once you have configured this file, you can get the information using the following command:

```
$ ansible-inventory --list -i scaleway_inventory.yml
```

The output will be:

```
{
    "_meta": {
        "hostvars": {
            "dd8e3ae9-0c7c-459e-bc7b-aba8bfa1bb8d": {
                "ansible_verbosity": 6,
                "arch": "x86_64",
                "commercial_type": "START1-S",
                "hostname": "foobar",
                "ipv4": "192.0.2.1",
                "organization": "00000000-1111-2222-3333-444444444444",
                "state": "running",
```

(continues on next page)

(continued from previous page)

```
                "tags": [
                    "web_server"
                ]
            }
        }
    },
    "all": {
        "children": [
            "ams1",
            "par1",
            "ungrouped",
            "web_server"
        ]
    },
    "ams1": {},
    "par1": {
        "hosts": [
            "dd8e3ae9-0c7c-459e-bc7b-aba8bfa1bb8d"
        ]
    },
    "ungrouped": {},
    "web_server": {
        "hosts": [
            "dd8e3ae9-0c7c-459e-bc7b-aba8bfa1bb8d"
        ]
    }
}
```

As you can see, we get different groups of hosts. `par1` and `ams1` are groups based on location. `web_server` is a group based on a tag.

In case a filter parameter is not defined, the plugin supposes all values possible are wanted. This means that for each tag that exists on your Scaleway compute nodes, a group based on each tag will be created.

Scaleway S3 object storage

Object Storage allows you to store any kind of objects (documents, images, videos, and so on). As the Scaleway API is S3 compatible, Ansible supports it natively through the modules: s3_bucket – Manage S3 buckets in AWS, DigitalOcean, Ceph, Walrus and FakeS3, aws_s3 – manage objects in S3.

You can find many examples in the scaleway_s3 integration tests.

```
- hosts: myserver
  vars:
    scaleway_region: nl-ams
    s3_url: https://s3.nl-ams.scw.cloud
  environment:
    # AWS_ACCESS_KEY matches your scaleway organization id available at https://cloud.
↪scaleway.com/#/account
    AWS_ACCESS_KEY: 00000000-1111-2222-3333-444444444444
    # AWS_SECRET_KEY matches a secret token that you can retrieve at https://cloud.
↪scaleway.com/#/credentials
```

(continues on next page)

(continued from previous page)

```
    AWS_SECRET_KEY: aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
  module_defaults:
    group/aws:
      s3_url: '{{ s3_url }}'
      region: '{{ scaleway_region }}'
  tasks:
   # use a fact instead of a variable, otherwise template is evaluate each time variable␣
↪is used
    - set_fact:
        bucket_name: "{{ 99999999 | random | to_uuid }}"

    # "requester_pays:" is mandatory because Scaleway doesn't implement related API
    # another way is to use aws_s3 and "mode: create" !
    - s3_bucket:
        name: '{{ bucket_name }}'
        requester_pays:

    - name: Another way to create the bucket
      aws_s3:
        bucket: '{{ bucket_name }}'
        mode: create
        encrypt: false
      register: bucket_creation_check

    - name: add something in the bucket
      aws_s3:
        mode: put
        bucket: '{{ bucket_name }}'
        src: /tmp/test.txt  #  needs to be created before
        object: test.txt
        encrypt: false  # server side encryption must be disabled
```

1.18.10 Vultr Guide

Ansible offers a set of modules to interact with Vultr cloud platform.

This set of module forms a framework that allows one to easily manage and orchestrate one's infrastructure on Vultr cloud platform.

Requirements

There is actually no technical requirement; simply an already created Vultr account.

Configuration

Vultr modules offer a rather flexible way with regard to configuration.

Configuration is read in that order:

- Environment Variables (eg. `VULTR_API_KEY`, `VULTR_API_TIMEOUT`)
- File specified by environment variable `VULTR_API_CONFIG`
- `vultr.ini` file located in current working directory
- `$HOME/.vultr.ini`

Ini file are structured this way:

```
[default]
key = MY_API_KEY
timeout = 60

[personal_account]
key = MY_PERSONAL_ACCOUNT_API_KEY
timeout = 30
```

If `VULTR_API_ACCOUNT` environment variable or `api_account` module parameter is not specified, modules will look for the section named "default".

Authentication

Before using the Ansible modules to interact with Vultr, you need an API key. If you don't yet own one, log in to Vultr go to Account, then API, enable API then the API key should show up.

Ensure you allow the usage of the API key from the proper IP addresses.

Refer to the Configuration section to find out where to put this information.

To check that everything is working properly run the following command:

```
#> VULTR_API_KEY=XXX ansible -m vultr_account_info localhost
localhost | SUCCESS => {
  "changed": false,
  "vultr_account_info": {
      "balance": -8.9,
      "last_payment_amount": -10.0,
      "last_payment_date": "2018-07-21 11:34:46",
      "pending_charges": 6.0
  },
  "vultr_api": {
      "api_account": "default",
      "api_endpoint": "https://api.vultr.com",
      "api_retries": 5,
      "api_timeout": 60
```

(continues on next page)

(continued from previous page)

```
  }
}
```

If a similar output displays then everything is setup properly, else please ensure the proper VULTR_API_KEY has been specified and that Access Controls on Vultr > Account > API page are accurate.

Usage

Since Vultr offers a public API, the execution of the module to manage the infrastructure on their platform will happen on localhost. This translates to:

```
---
- hosts: localhost
  tasks:
    - name: Create a 10G volume
      vultr_block_storage:
        name: my_disk
        size: 10
        region: New Jersey
```

From that point on, only your creativity is the limit. Make sure to read the documentation of the available modules.

Dynamic Inventory

Ansible provides a dynamic inventory plugin for Vultr. The configuration process is exactly the same as for the modules.

To be able to use it you need to enable it first by specifying the following in the ansible.cfg file:

```
[inventory]
enable_plugins=vultr
```

And provide a configuration file to be used with the plugin, the minimal configuration file looks like this:

```
---
plugin: vultr
```

To list the available hosts one can simply run:

```
#> ansible-inventory -i vultr.yml --list
```

For example, this allows you to take action on nodes grouped by location or OS name:

```
---
- hosts: Amsterdam
  tasks:
    - name: Rebooting the machine
      shell: reboot
      become: True
```

Integration tests

Ansible includes integration tests for all Vultr modules.

These tests are meant to run against the public Vultr API and that is why they require a valid key to access the API.

Prepare the test setup:

```
$ cd ansible # location the ansible source is
$ source ./hacking/env-setup
```

Set the Vultr API key:

```
$ cd test/integration
$ cp cloud-config-vultr.ini.template cloud-config-vultr.ini
$ vi cloud-config-vultr.ini
```

Run all Vultr tests:

```
$ ansible-test integration cloud/vultr/ -v --diff --allow-unsupported
```

To run a specific test, for example vultr_account_info:

```
$ ansible-test integration cloud/vultr/vultr_account_info -v --diff --allow-unsupported
```

1.19 Network Technology Guides

The guides in this section cover using Ansible with specific network technologies. They explore particular use cases in greater depth and provide a more "top-down" explanation of some basic features.

1.19.1 Cisco ACI Guide

What is Cisco ACI ?

Application Centric Infrastructure (ACI)

The Cisco Application Centric Infrastructure (ACI) allows application requirements to define the network. This architecture simplifies, optimizes, and accelerates the entire application deployment life cycle.

Application Policy Infrastructure Controller (APIC)

The APIC manages the scalable ACI multi-tenant fabric. The APIC provides a unified point of automation and management, policy programming, application deployment, and health monitoring for the fabric. The APIC, which is implemented as a replicated synchronized clustered controller, optimizes performance, supports any application anywhere, and provides unified operation of the physical and virtual infrastructure.

The APIC enables network administrators to easily define the optimal network for applications. Data center operators can clearly see how applications consume network resources, easily isolate and troubleshoot application and infrastructure problems, and monitor and profile resource usage patterns.

The Cisco Application Policy Infrastructure Controller (APIC) API enables applications to directly connect with a secure, shared, high-performance resource pool that includes network, compute, and storage capabilities.

ACI Fabric

The Cisco Application Centric Infrastructure (ACI) Fabric includes Cisco Nexus 9000 Series switches with the APIC to run in the leaf/spine ACI fabric mode. These switches form a "fat-tree" network by connecting each leaf node to each spine node; all other devices connect to the leaf nodes. The APIC manages the ACI fabric.

The ACI fabric provides consistent low-latency forwarding across high-bandwidth links (40 Gbps, with a 100-Gbps future capability). Traffic with the source and destination on the same leaf switch is handled locally, and all other traffic travels from the ingress leaf to the egress leaf through a spine switch. Although this architecture appears as two hops from a physical perspective, it is actually a single Layer 3 hop because the fabric operates as a single Layer 3 switch.

The ACI fabric object-oriented operating system (OS) runs on each Cisco Nexus 9000 Series node. It enables programming of objects for each configurable element of the system. The ACI fabric OS renders policies from the APIC into a concrete model that runs in the physical infrastructure. The concrete model is analogous to compiled software; it is the form of the model that the switch operating system can execute.

All the switch nodes contain a complete copy of the concrete model. When an administrator creates a policy in the APIC that represents a configuration, the APIC updates the logical model. The APIC then performs the intermediate step of creating a fully elaborated policy that it pushes into all the switch nodes where the concrete model is updated.

The APIC is responsible for fabric activation, switch firmware management, network policy configuration, and instantiation. While the APIC acts as the centralized policy and network management engine for the fabric, it is completely removed from the data path, including the forwarding topology. Therefore, the fabric can still forward traffic even when communication with the APIC is lost.

More information

Various resources exist to start learning ACI, here is a list of interesting articles from the community.

- Adam Raffe: Learning ACI
- Luca Relandini: ACI for dummies
- Cisco DevNet Learning Labs about ACI

Using the ACI modules

The Ansible ACI modules provide a user-friendly interface to managing your ACI environment using Ansible playbooks.

For instance ensuring that a specific tenant exists, is done using the following Ansible task using the aci_tenant module:

```yaml
- name: Ensure tenant customer-xyz exists
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password

    tenant: customer-xyz
    description: Customer XYZ
    state: present
```

A complete list of existing ACI modules is available on the content tab of the ACI collection on Ansible Galaxy.

If you want to learn how to write your own ACI modules to contribute, look at the Developing Cisco ACI modules section.

Querying ACI configuration

A module can also be used to query a specific object.

```
- name: Query tenant customer-xyz
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password

    tenant: customer-xyz
    state: query
  register: my_tenant
```

Or query all objects.

```
- name: Query all tenants
  aci_tenant:
    host: my-apic-1
    username: admin
    password: my-password

    state: query
  register: all_tenants
```

After registering the return values of the aci_tenant task as shown above, you can access all tenant information from variable `all_tenants`.

Running on the controller locally

As originally designed, Ansible modules are shipped to and run on the remote target(s), however the ACI modules (like most network-related modules) do not run on the network devices or controller (in this case the APIC), but they talk directly to the APIC's REST interface.

For this very reason, the modules need to run on the local Ansible controller (or are delegated to another system that *can* connect to the APIC).

Gathering facts

Because we run the modules on the Ansible controller gathering facts will not work. That is why when using these ACI modules it is mandatory to disable facts gathering. You can do this globally in your `ansible.cfg` or by adding `gather_facts:  false` to every play.

```
 - name: Another play in my playbook
   hosts: my-apic-1
   gather_facts: false
   tasks:
   - name: Create a tenant
     aci_tenant:
       ...
```

Delegating to localhost

So let us assume we have our target configured in the inventory using the FQDN name as the `ansible_host` value, as shown below.

```
apics:
  my-apic-1:
    ansible_host: apic01.fqdn.intra
    ansible_user: admin
    ansible_password: my-password
```

One way to set this up is to add to every task the directive: `delegate_to:  localhost`.

```
- name: Query all tenants
  aci_tenant:
    host: '{{ ansible_host }}'
    username: '{{ ansible_user }}'
    password: '{{ ansible_password }}'

    state: query
  delegate_to: localhost
  register: all_tenants
```

If one would forget to add this directive, Ansible will attempt to connect to the APIC using SSH and attempt to copy the module and run it remotely. This will fail with a clear error, yet may be confusing to some.

Using the local connection method

Another option frequently used, is to tie the `local` connection method to this target so that every subsequent task for this target will use the local connection method (hence run it locally, rather than use SSH).

In this case the inventory may look like this:

```
apics:
  my-apic-1:
    ansible_host: apic01.fqdn.intra
    ansible_user: admin
    ansible_password: my-password
    ansible_connection: local
```

But used tasks do not need anything special added.

```
- name: Query all tenants
  aci_tenant:
    host: '{{ ansible_host }}'
    username: '{{ ansible_user }}'
    password: '{{ ansible_password }}'

    state: query
  register: all_tenants
```

Hint: For clarity we have added `delegate_to:  localhost` to all the examples in the module documentation. This helps to ensure first-time users can easily copy&paste parts and make them work with a minimum of effort.

Common parameters

Every Ansible ACI module accepts the following parameters that influence the module's communication with the APIC REST API:

host
: Hostname or IP address of the APIC.

port
: Port to use for communication. (Defaults to `443` for HTTPS, and `80` for HTTP)

username
: User name used to log on to the APIC. (Defaults to `admin`)

password
: Password for `username` to log on to the APIC, using password-based authentication.

private_key
: Private key for `username` to log on to APIC, using signature-based authentication. This could either be the raw private key content (include header/footer) or a file that stores the key content. *New in version 2.5*

certificate_name
: Name of the certificate in the ACI Web GUI. This defaults to either the `username` value or the `private_key` file base name). *New in version 2.5*

timeout
: Timeout value for socket-level communication.

use_proxy
: Use system proxy settings. (Defaults to `yes`)

use_ssl
: Use HTTPS or HTTP for APIC REST communication. (Defaults to `yes`)

validate_certs
: Validate certificate when using HTTPS communication. (Defaults to `yes`)

output_level
: Influence the level of detail ACI modules return to the user. (One of `normal`, `info` or `debug`) *New in version 2.5*

Proxy support

By default, if an environment variable `<protocol>_proxy` is set on the target host, requests will be sent through that proxy. This behaviour can be overridden by setting a variable for this task (see *Setting the remote environment*), or by using the `use_proxy` module parameter.

HTTP redirects can redirect from HTTP to HTTPS so ensure that the proxy environment for both protocols is correctly configured.

If proxy support is not needed, but the system may have it configured nevertheless, use the parameter `use_proxy: false` to avoid accidental system proxy usage.

Hint: Selective proxy support using the `no_proxy` environment variable is also supported.

Return values

New in version 2.5.

The following values are always returned:

current
: The resulting state of the managed object, or results of your query.

The following values are returned when `output_level: info`:

previous
: The original state of the managed object (before any change was made).

proposed
: The proposed config payload, based on user-supplied values.

sent
: The sent config payload, based on user-supplied values and the existing configuration.

The following values are returned when `output_level: debug` or `ANSIBLE_DEBUG=1`:

filter_string
: The filter used for specific APIC queries.

method
: The HTTP method used for the sent payload. (Either `GET` for queries, `DELETE` or `POST` for changes)

response
: The HTTP response from the APIC.

status
: The HTTP status code for the request.

url
: The url used for the request.

Note: The module return values are documented in detail as part of each module's documentation.

More information

Various resources exist to start learn more about ACI programmability, we recommend the following links:

- Developing Cisco ACI modules
- Jacob McGill: Automating Cisco ACI with Ansible
- Cisco DevNet Learning Labs about ACI and Ansible

ACI authentication

Password-based authentication

If you want to log on using a username and password, you can use the following parameters with your ACI modules:

```
username: admin
password: my-password
```

Password-based authentication is very simple to work with, but it is not the most efficient form of authentication from ACI's point-of-view as it requires a separate login-request and an open session to work. To avoid having your session time-out and requiring another login, you can use the more efficient Signature-based authentication.

Note: Password-based authentication also may trigger anti-DoS measures in ACI v3.1+ that causes session throttling and results in HTTP 503 errors and login failures.

Warning: Never store passwords in plain text.

The "Vault" feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See *Using encrypted variables and files* for more information.

Signature-based authentication using certificates

New in version 2.5.

Using signature-based authentication is more efficient and more reliable than password-based authentication.

Generate certificate and private key

Signature-based authentication requires a (self-signed) X.509 certificate with private key, and a configuration step for your AAA user in ACI. To generate a working X.509 certificate and private key, use the following procedure:

```
$ openssl req -new -newkey rsa:1024 -days 36500 -nodes -x509 -keyout admin.key -out␣
↪admin.crt -subj '/CN=Admin/O=Your Company/C=US'
```

Configure your local user

Perform the following steps:

- Add the X.509 certificate to your ACI AAA local user at *ADMIN » AAA*
- Click *AAA Authentication*
- Check that in the *Authentication* field the *Realm* field displays *Local*
- Expand *Security Management » Local Users*
- Click the name of the user you want to add a certificate to, in the *User Certificates* area
- Click the + sign and in the *Create X509 Certificate* enter a certificate name in the *Name* field

  - If you use the basename of your private key here, you don't need to enter `certificate_name` in Ansible

- Copy and paste your X.509 certificate in the *Data* field.

You can automate this by using the following Ansible task:

```
- name: Ensure we have a certificate installed
  aci_aaa_user_certificate:
    host: my-apic-1
    username: admin
    password: my-password

    aaa_user: admin
    certificate_name: admin
    certificate: "{{ lookup('file', 'pki/admin.crt') }}"  # This will read the␣
↪certificate data from a local file
```

Note: Signature-based authentication only works with local users.

Use signature-based authentication with Ansible

You need the following parameters with your ACI module(s) for it to work:

```
username: admin
private_key: pki/admin.key
certificate_name: admin  # This could be left out !
```

or you can use the private key content:

```
username: admin
private_key: |
    -----BEGIN PRIVATE KEY-----
    <<your private key content>>
    -----END PRIVATE KEY-----
certificate_name: admin  # This could be left out !
```

Hint: If you use a certificate name in ACI that matches the private key's basename, you can leave out the `certificate_name` parameter like the example above.

Using Ansible Vault to encrypt the private key

New in version 2.8.

To start, encrypt the private key and give it a strong password.

```
ansible-vault encrypt admin.key
```

Use a text editor to open the private-key. You should have an encrypted cert now.

```
$ANSIBLE_VAULT;1.1;AES256
56484318584354658465121889743213151843149454864654151618131547984132165489484654
45641818198456456489479874513215489484843614848456466655432455488484654848489498
....
```

Copy and paste the new encrypted cert into your playbook as a new variable.

```
private_key: !vault |
      $ANSIBLE_VAULT;1.1;AES256
      56484318584354658465121889743213151843149454864654151618131547984132165489484654
      45641818198456456489479874513215489484843614848456466655432455488484654848489498
      ....
```

Use the new variable for the private_key:

```
username: admin
private_key: "{{ private_key }}"
certificate_name: admin  # This could be left out !
```

When running the playbook, use "–ask-vault-pass" to decrypt the private key.

```
ansible-playbook site.yaml --ask-vault-pass
```

More information

- Detailed information about Signature-based Authentication is available from Cisco APIC Signature-Based Transactions.
- More information on Ansible Vault can be found on the *Ansible Vault* page.

Using ACI REST with Ansible

While already a lot of ACI modules exists in the Ansible distribution, and the most common actions can be performed with these existing modules, there's always something that may not be possible with off-the-shelf modules.

The aci_rest module provides you with direct access to the APIC REST API and enables you to perform any task not already covered by the existing modules. This may seem like a complex undertaking, but you can generate the needed REST payload for any action performed in the ACI web interface effortlessly.

Built-in idempotency

Because the APIC REST API is intrinsically idempotent and can report whether a change was made, the aci_rest module automatically inherits both capabilities and is a first-class solution for automating your ACI infrastructure. As a result, users that require more powerful low-level access to their ACI infrastructure don't have to give up on idempotency and don't have to guess whether a change was performed when using the aci_rest module.

Using the aci_rest module

The aci_rest module accepts the native XML and JSON payloads, but additionally accepts inline YAML payload (structured like JSON). The XML payload requires you to use a path ending with `.xml` whereas JSON or YAML require the path to end with `.json`.

When you're making modifications, you can use the POST or DELETE methods, whereas doing just queries require the GET method.

For instance, if you would like to ensure a specific tenant exists on ACI, these below four examples are functionally identical:

XML (Native ACI REST)

```
- aci_rest:
    host: my-apic-1
    private_key: pki/admin.key

    method: post
    path: /api/mo/uni.xml
    content: |
      <fvTenant name="customer-xyz" descr="Customer XYZ"/>
```

JSON (Native ACI REST)

```
- aci_rest:
    host: my-apic-1
    private_key: pki/admin.key

    method: post
    path: /api/mo/uni.json
    content:
      {
        "fvTenant": {
          "attributes": {
            "name": "customer-xyz",
            "descr": "Customer XYZ"
          }
        }
      }
```

YAML (Ansible-style REST)

```
- aci_rest:
    host: my-apic-1
    private_key: pki/admin.key

    method: post
    path: /api/mo/uni.json
    content:
      fvTenant:
        attributes:
          name: customer-xyz
          descr: Customer XYZ
```

Ansible task (Dedicated module)

```yaml
- aci_tenant:
    host: my-apic-1
    private_key: pki/admin.key

    tenant: customer-xyz
    description: Customer XYZ
    state: present
```

Hint: The XML format is more practical when there is a need to template the REST payload (inline), but the YAML format is more convenient for maintaining your infrastructure-as-code and feels more naturally integrated with Ansible playbooks. The dedicated modules offer a more simple, abstracted, but also a more limited experience. Use what feels best for your use-case.

More information

Plenty of resources exist to learn about ACI's APIC REST interface, we recommend the links below:

- The ACI collection on Ansible Galaxy
- APIC REST API Configuration Guide – Detailed guide on how the APIC REST API is designed and used, incl. many examples
- APIC Management Information Model reference – Complete reference of the APIC object model
- Cisco DevNet Learning Labs about ACI and REST

Operational examples

Here is a small overview of useful operational tasks to reuse in your playbooks.

Feel free to contribute more useful snippets.

Waiting for all controllers to be ready

You can use the below task after you started to build your APICs and configured the cluster to wait until all the APICs have come online. It will wait until the number of controllers equals the number listed in the `apic` inventory group.

```yaml
- name: Waiting for all controllers to be ready
  aci_rest:
    host: my-apic-1
    private_key: pki/admin.key
    method: get
    path: /api/node/class/topSystem.json?query-target-filter=eq(topSystem.role,
↪"controller")
  register: topsystem
  until: topsystem|success and topsystem.totalCount|int >= groups['apic']|count >= 3
  retries: 20
  delay: 30
```

Waiting for cluster to be fully-fit

The below example waits until the cluster is fully-fit. In this example you know the number of APICs in the cluster and you verify each APIC reports a 'fully-fit' status.

```
- name: Waiting for cluster to be fully-fit
  aci_rest:
    host: my-apic-1
    private_key: pki/admin.key
    method: get
    path: /api/node/class/infraWiNode.json?query-target-filter=wcard(infraWiNode.dn,
↪"topology/pod-1/node-1/av")
  register: infrawinode
  until: >
    infrawinode|success and
    infrawinode.totalCount|int >= groups['apic']|count >= 3 and
    infrawinode.imdata[0].infraWiNode.attributes.health == 'fully-fit' and
    infrawinode.imdata[1].infraWiNode.attributes.health == 'fully-fit' and
    infrawinode.imdata[2].infraWiNode.attributes.health == 'fully-fit'
  retries: 30
  delay: 30
```

APIC error messages

The following error messages may occur and this section can help you understand what exactly is going on and how to fix/avoid them.

APIC Error 122: unknown managed object class 'polUni'
In case you receive this error while you are certain your aci_rest payload and object classes are seemingly correct, the issue might be that your payload is not in fact correct JSON (for example, the sent payload is using single quotes, rather than double quotes), and as a result the APIC is not correctly parsing your object classes from the payload. One way to avoid this is by using a YAML or an XML formatted payload, which are easier to construct correctly and modify later.

APIC Error 400: invalid data at line '1'. Attributes are missing, tag 'attributes' must be specified first, before any other tag
Although the JSON specification allows unordered elements, the APIC REST API requires that the JSON `attributes` element precede the `children` array or other elements. So you need to ensure that your payload conforms to this requirement. Sorting your dictionary keys will do the trick just fine. If you don't have any attributes, it may be necessary to add: `attributes: {}` as the APIC does expect the entry to precede any `children`.

APIC Error 801: property descr of uni/tn-TENANT/ap-AP failed validation for value 'A "legacy" network'
Some values in the APIC have strict format-rules to comply to, and the internal APIC validation check for the provided value failed. In the above case, the `description` parameter (internally known as `descr`) only accepts values conforming to Regex: `[a-zA-Z0-9\\!#$%()*,-./:;@ _{|}~?&+]+`, in general it must not include quotes or square brackets.

Known issues

The aci_rest module is a wrapper around the APIC REST API. As a result any issues related to the APIC will be reflected in the use of this module.

All below issues either have been reported to the vendor, and most can simply be avoided.

Too many consecutive API calls may result in connection throttling
: Starting with ACI v3.1 the APIC will actively throttle password-based authenticated connection rates over a specific threshold. This is as part of an anti-DDOS measure but can act up when using Ansible with ACI using password-based authentication. Currently, one solution is to increase this threshold within the nginx configuration, but using signature-based authentication is recommended.

  NOTE: It is advisable to use signature-based authentication with ACI as it not only prevents connection-throttling, but also improves general performance when using the ACI modules.

Specific requests may not reflect changes correctly (#35401)
: There is a known issue where specific requests to the APIC do not properly reflect changed in the resulting output, even when we request those changes explicitly from the APIC. In one instance using the path `api/node/mo/uni/infra.xml` fails, where `api/node/mo/uni/infra/.xml` does work correctly.

  NOTE: A workaround is to register the task return values (for example, `register:  this`) and influence when the task should report a change by adding: `changed_when:  this.imdata != []`.

Specific requests are known to not be idempotent (#35050)
: The behaviour of the APIC is inconsistent to the use of `status="created"` and `status="deleted"`. The result is that when you use `status="created"` in your payload the resulting tasks are not idempotent and creation will fail when the object was already created. However this is not the case with `status="deleted"` where such call to an non-existing object does not cause any failure whatsoever.

  NOTE: A workaround is to avoid using `status="created"` and instead use `status="modified"` when idempotency is essential to your workflow..

Setting user password is not idempotent (#35544)
: Due to an inconsistency in the APIC REST API, a task that sets the password of a locally-authenticated user is not idempotent. The APIC will complain with message `Password history check:  user dag should not use previous 5 passwords`.

  NOTE: There is no workaround for this issue.

ACI Ansible community

If you have specific issues with the ACI modules, or a feature request, or you like to contribute to the ACI project by proposing changes or documentation updates, look at the Ansible Community wiki ACI page at: https://github.com/ansible/community/wiki/Network:-ACI

You will find our roadmap, an overview of open ACI issues and pull-requests, and more information about who we are. If you have an interest in using ACI with Ansible, feel free to join! We occasionally meet online (on the #ansible-network chat channel, using Matrix at ansible.im or using IRC at irc.libera.chat) to track progress and prepare for new Ansible releases.

See also:

ACI collection on Ansible Galaxy
: View the content tab for a complete list of supported ACI modules.

Developing Cisco ACI modules
A walkthrough on how to develop new Cisco ACI modules to contribute back.

ACI community
The Ansible ACI community wiki page, includes roadmap, ideas and development documentation.

network_guide
A detailed guide on how to use Ansible for automating network infrastructure.

Network Working Group
The Ansible Network community page, includes contact information and meeting information.

User Mailing List
Have a question? Stop by the google group!

1.19.2 Cisco Meraki Guide

What is Cisco Meraki?

Cisco Meraki is an easy-to-use, cloud-based, network infrastructure platform for enterprise environments. While most network hardware uses command-line interfaces (CLIs) for configuration, Meraki uses an easy-to-use Dashboard hosted in the Meraki cloud. No on-premises management hardware or software is required - only the network infrastructure to run your business.

MS Switches

Meraki MS switches come in multiple flavors and form factors. Meraki switches support 10/100/1000/10000 ports, as well as Cisco's mGig technology for 2.5/5/10Gbps copper connectivity. 8, 24, and 48 port flavors are available with PoE (802.3af/802.3at/UPoE) available on many models.

MX Firewalls

Meraki's MX firewalls support full layer 3-7 deep packet inspection. MX firewalls are compatible with a variety of VPN technologies including IPSec, SSL VPN, and Meraki's easy-to-use AutoVPN.

MR Wireless Access Points

MR access points are enterprise-class, high-performance access points for the enterprise. MR access points have MIMO technology and integrated beamforming built-in for high performance applications. BLE allows for advanced location applications to be developed with no on-premises analytics platforms.

Using the Meraki modules

Meraki modules provide a user-friendly interface to manage your Meraki environment using Ansible. For example, details about SNMP settings for a particular organization can be discovered using the module *meraki_snmp <meraki_snmp_module>*.

```
- name: Query SNMP settings
  meraki_snmp:
    api_key: abc123
    org_name: AcmeCorp
    state: query
  delegate_to: localhost
```

Information about a particular object can be queried. For example, the *meraki_admin <meraki_admin_module>* module supports

```
- name: Gather information about Jane Doe
  meraki_admin:
    api_key: abc123
    org_name: AcmeCorp
    state: query
    email: janedoe@email.com
  delegate_to: localhost
```

Common Parameters

All Ansible Meraki modules support the following parameters which affect communication with the Meraki Dashboard API. Most of these should only be used by Meraki developers and not the general public.

host
: Hostname or IP of Meraki Dashboard.

use_https
: Specifies whether communication should be over HTTPS. (Defaults to `yes`)

use_proxy
: Whether to use a proxy for any communication.

validate_certs
: Determine whether certificates should be validated or trusted. (Defaults to `yes`)

These are the common parameters which are used for most every module.

org_name
: Name of organization to perform actions in.

org_id
: ID of organization to perform actions in.

net_name
: Name of network to perform actions in.

net_id
: ID of network to perform actions in.

state
: General specification of what action to take. `query` does lookups. `present` creates or edits. `absent` deletes.

Hint: Use the `org_id` and `net_id` parameters when possible. `org_name` and `net_name` require additional behind-the-scenes API calls to learn the ID values. `org_id` and `net_id` will perform faster.

Meraki Authentication

All API access with the Meraki Dashboard requires an API key. An API key can be generated from the organization's settings page. Each play in a playbook requires the `api_key` parameter to be specified.

The "Vault" feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See *Using encrypted variables and files* for more information.

Meraki's API returns a 404 error if the API key is not correct. It does not provide any specific error saying the key is incorrect. If you receive a 404 error, check the API key first.

Returned Data Structures

Meraki and its related Ansible modules return most information in the form of a list. For example, this is returned information by `meraki_admin` querying administrators. It returns a list even though there's only one.

```
[
    {
        "orgAccess": "full",
        "name": "John Doe",
        "tags": [],
        "networks": [],
        "email": "john@doe.com",
        "id": "12345677890"
    }
]
```

Handling Returned Data

Since Meraki's response data uses lists instead of properly keyed dictionaries for responses, certain strategies should be used when querying data for particular information. For many situations, use the `selectattr()` Jinja2 function.

Merging Existing and New Data

Ansible's Meraki modules do not allow for manipulating data. For example, you may need to insert a rule in the middle of a firewall ruleset. Ansible and the Meraki modules lack a way to directly merge to manipulate data. However, a playlist can use a few tasks to split the list where you need to insert a rule and then merge them together again with the new rule added. The steps involved are as follows:

1. **Create blank "front" and "back" lists.**

```
vars:
  - front_rules: []
  - back_rules: []
```

2. **Get existing firewall rules from Meraki and create a new variable.**

```
- name: Get firewall rules
  meraki_mx_l3_firewall:
    auth_key: abc123
    org_name: YourOrg
    net_name: YourNet
    state: query
  delegate_to: localhost
  register: rules
- set_fact:
    original_ruleset: '{{rules.data}}'
```

3. **Write the new rule. The new rule needs to be in a list so it can be merged with other lists in an upcoming step. The blank - puts the rule in a list so it can be merged.**

```
- set_fact:
    new_rule:
      -
        - comment: Block traffic to server
          src_cidr: 192.0.1.0/24
          src_port: any
          dst_cidr: 192.0.1.2/32
          dst_port: any
          protocol: any
          policy: deny
```

4. **Split the rules into two lists. This assumes the existing ruleset is 2 rules long.**

```
- set_fact:
    front_rules: '{{front_rules + [ original_ruleset[:1] ]}}'
- set_fact:
    back_rules: '{{back_rules + [ original_ruleset[1:] ]}}'
```

5. **Merge rules with the new rule in the middle.**

```
- set_fact:
    new_ruleset: '{{front_rules + new_rule + back_rules}}'
```

6. **Upload new ruleset to Meraki.**

```
- name: Set two firewall rules
  meraki_mx_l3_firewall:
    auth_key: abc123
    org_name: YourOrg
    net_name: YourNet
    state: present
    rules: '{{ new_ruleset }}'
  delegate_to: localhost
```

Error Handling

Ansible's Meraki modules will often fail if improper or incompatible parameters are specified. However, there will likely be scenarios where the module accepts the information but the Meraki API rejects the data. If this happens, the error will be returned in the body field for HTTP status of 400 return code.

Meraki's API returns a 404 error if the API key is not correct. It does not provide any specific error saying the key is incorrect. If you receive a 404 error, check the API key first. 404 errors can also occur if improper object IDs (ex. `org_id`) are specified.

1.19.3 Infoblox Guide

Topics

This guide describes how to use Ansible with the Infoblox Network Identity Operating System (NIOS). With Ansible integration, you can use Ansible playbooks to automate Infoblox Core Network Services for IP address management (IPAM), DNS, and inventory tracking.

You can review simple example tasks in the documentation for any of the NIOS modules or look at the *Use cases with modules* section for more elaborate examples. See the Infoblox website for more information on the Infoblox product.

Note: You can retrieve most of the example playbooks used in this guide from the network-automation/infoblox_ansible GitHub repository.

Prerequisites

Before using Ansible nios modules with Infoblox, you must install the infoblox-client on your Ansible control node:

```
$ sudo pip install infoblox-client
```

Note: You need an NIOS account with the WAPI feature enabled to use Ansible with Infoblox.

Credentials and authenticating

To use Infoblox nios modules in playbooks, you need to configure the credentials to access your Infoblox system. The examples in this guide use credentials stored in <playbookdir>/group_vars/nios.yml. Replace these values with your Infoblox credentials:

```
---
nios_provider:
  host: 192.0.0.2
  username: admin
  password: ansible
```

NIOS lookup plugins

Ansible includes the following lookup plugins for NIOS:

- nios Uses the Infoblox WAPI API to fetch NIOS specified objects, for example network views, DNS views, and host records.
- nios_next_ip Provides the next available IP address from a network. You'll see an example of this in *Creating a host record*.
- nios_next_network - Returns the next available network range for a network-container.

You must run the NIOS lookup plugins locally by specifying `connection: local`. See *lookup plugins* for more detail.

Retrieving all network views

To retrieve all network views and save them in a variable, use the set_fact module with the nios lookup plugin:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: fetch all networkview objects
      set_fact:
        networkviews: "{{ lookup('nios', 'networkview', provider=nios_provider) }}"

    - name: check the networkviews
      debug:
        var: networkviews
```

Retrieving a host record

To retrieve a set of host records, use the `set_fact` module with the `nios` lookup plugin and include a filter for the specific hosts you want to retrieve:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: fetch host leaf01
      set_fact:
         host: "{{ lookup('nios', 'record:host', filter={'name': 'leaf01.ansible.com'},␣
↪provider=nios_provider) }}"

    - name: check the leaf01 return variable
      debug:
        var: host

    - name: debug specific variable (ipv4 address)
      debug:
        var: host.ipv4addrs[0].ipv4addr

    - name: fetch host leaf02
      set_fact:
        host: "{{ lookup('nios', 'record:host', filter={'name': 'leaf02.ansible.com'},␣
↪provider=nios_provider) }}"

    - name: check the leaf02 return variable
      debug:
        var: host
```

If you run this `get_host_record.yml` playbook, you should see results similar to the following:

```
$ ansible-playbook get_host_record.yml

PLAY [localhost]␣
```

(continues on next page)

(continued from previous page)

```
↪***************************************************************************************

TASK [fetch host leaf01]␣
↪******************************************************************************
ok: [localhost]

TASK [check the leaf01 return variable]␣
↪*************************************************************
ok: [localhost] => {
< ...output shortened...>
    "host": {
        "ipv4addrs": [
            {
                "configure_for_dhcp": false,
                "host": "leaf01.ansible.com",
            }
        ],
        "name": "leaf01.ansible.com",
        "view": "default"
    }
}

TASK [debug specific variable (ipv4 address)]␣
↪******************************************************
ok: [localhost] => {
    "host.ipv4addrs[0].ipv4addr": "192.168.1.11"
}

TASK [fetch host leaf02]␣
↪******************************************************************************
ok: [localhost]

TASK [check the leaf02 return variable]␣
↪*************************************************************
ok: [localhost] => {
< ...output shortened...>
    "host": {
        "ipv4addrs": [
            {
                "configure_for_dhcp": false,
                "host": "leaf02.example.com",
                "ipv4addr": "192.168.1.12"
            }
        ],
    }
}

PLAY RECAP␣
↪******************************************************************************************
localhost                  : ok=5    changed=0    unreachable=0    failed=0
```

The output above shows the host record for `leaf01.ansible.com` and `leaf02.ansible.com` that were retrieved by the `nios` lookup plugin. This playbook saves the information in variables which you can use in other playbooks. This allows you to use Infoblox as a single source of truth to gather and use information that changes dynamically. See *Using Variables* for more information on using Ansible variables. See the nios examples for more data options that you can retrieve.

You can access these playbooks at Infoblox lookup playbooks.

Use cases with modules

You can use the `nios` modules in tasks to simplify common Infoblox workflows. Be sure to set up your *NIOS credentials* before following these examples.

Configuring an IPv4 network

To configure an IPv4 network, use the nios_network module:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: Create a network on the default network view
      nios_network:
        network: 192.168.100.0/24
        comment: sets the IPv4 network
        options:
          - name: domain-name
            value: ansible.com
        state: present
        provider: "{{nios_provider}}"
```

Notice the last parameter, `provider`, uses the variable `nios_provider` defined in the `group_vars/` directory.

Creating a host record

To create a host record named *leaf03.ansible.com* on the newly-created IPv4 network:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: configure an IPv4 host record
      nios_host_record:
        name: leaf03.ansible.com
        ipv4addrs:
          - ipv4addr:
              "{{ lookup('nios_next_ip', '192.168.100.0/24', provider=nios_provider)[0] }
↪}"
        state: present
provider: "{{nios_provider}}"
```

Notice the IPv4 address in this example uses the nios_next_ip lookup plugin to find the next available IPv4 address on the network.

Creating a forward DNS zone

To configure a forward DNS zone use, the `nios_zone` module:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: Create a forward DNS zone called ansible-test.com
      nios_zone:
        name: ansible-test.com
        comment: local DNS zone
        state: present
        provider: "{{ nios_provider }}"
```

Creating a reverse DNS zone

To configure a reverse DNS zone:

```
---
- hosts: nios
  connection: local
  tasks:
    - name: configure a reverse mapping zone on the system using IPV6 zone format
      nios_zone:
        name: 100::1/128
        zone_format: IPV6
        state: present
        provider: "{{ nios_provider }}"
```

Dynamic inventory script

You can use the Infoblox dynamic inventory script to import your network node inventory with Infoblox NIOS. To gather the inventory from Infoblox, you need two files:

- infoblox.yaml - A file that specifies the NIOS provider arguments and optional filters.
- infoblox.py - The python script that retrieves the NIOS inventory.

Note: Please note that the inventory script only works when Ansible 2.9, 2.10 or 3 have been installed. The inventory script will eventually be removed from community.general, and will not work if *community.general* is only installed with *ansible-galaxy collection install*. Please use the inventory plugin from infoblox.nios_modules instead.

To use the Infoblox dynamic inventory script:

1. Download the `infoblox.yaml` file and save it in the `/etc/ansible` directory.
2. Modify the `infoblox.yaml` file with your NIOS credentials.
3. Download the `infoblox.py` file and save it in the `/etc/ansible/hosts` directory.
4. Change the permissions on the `infoblox.py` file to make the file an executable:

```
$ sudo chmod +x /etc/ansible/hosts/infoblox.py
```

You can optionally use `./infoblox.py --list` to test the script. After a few minutes, you should see your Infoblox inventory in JSON format. You can explicitly use the Infoblox dynamic inventory script as follows:

```
$ ansible -i infoblox.py all -m ping
```

You can also implicitly use the Infoblox dynamic inventory script by including it in your inventory directory (`etc/ansible/hosts` by default). See *Working with dynamic inventory* for more details.

See also:

Infoblox website
The Infoblox website

Infoblox and Ansible Deployment Guide
The deployment guide for Ansible integration provided by Infoblox.

Infoblox Integration in Ansible 2.5
Ansible blog post about Infoblox.

Ansible NIOS modules
The list of supported NIOS modules, with examples.

Infoblox Ansible Examples
Infoblox example playbooks.

To learn more about Network Automation with Ansible, see *Network Getting Started* and *Network Advanced Topics*.

1.20 Virtualization and Containerization Guides

The guides in this section cover integrating Ansible with popular tools for creating virtual machines and containers. They explore particular use cases in greater depth and provide a more "top-down" explanation of some basic features.

1.20.1 Docker Guide

The content on this page has moved. Please see the updated Docker Guide in the community.docker collection.

1.20.2 Vagrant Guide

Introduction

Vagrant is a tool to manage virtual machine environments, and allows you to configure and use reproducible work environments on top of various virtualization and cloud platforms. It also has integration with Ansible as a provisioner for these virtual machines, and the two tools work together well.

This guide will describe how to use Vagrant 1.7+ and Ansible together.

If you're not familiar with Vagrant, you should visit the documentation.

This guide assumes that you already have Ansible installed and working. Running from a Git checkout is fine. Follow the *Installing Ansible* guide for more information.

Vagrant Setup

The first step once you've installed Vagrant is to create a `Vagrantfile` and customize it to suit your needs. This is covered in detail in the Vagrant documentation, but here is a quick example that includes a section to use the Ansible provisioner to manage a single machine:

```
# This guide is optimized for Vagrant 1.8 and above.
# Older versions of Vagrant put less info in the inventory they generate.
Vagrant.require_version ">= 1.8.0"

Vagrant.configure(2) do |config|

  config.vm.box = "ubuntu/bionic64"

  config.vm.provision "ansible" do |ansible|
    ansible.verbose = "v"
    ansible.playbook = "playbook.yml"
  end
end
```

Notice the `config.vm.provision` section that refers to an Ansible playbook called `playbook.yml` in the same directory as the `Vagrantfile`. Vagrant runs the provisioner once the virtual machine has booted and is ready for SSH access.

There are a lot of Ansible options you can configure in your `Vagrantfile`. Visit the Ansible Provisioner documentation for more information.

```
$ vagrant up
```

This will start the VM, and run the provisioning playbook (on the first VM startup).

To re-run a playbook on an existing VM, just run:

```
$ vagrant provision
```

This will re-run the playbook against the existing VM.

Note that having the `ansible.verbose` option enabled will instruct Vagrant to show the full `ansible-playbook` command used behind the scene, as illustrated by this example:

```
$ PYTHONUNBUFFERED=1 ANSIBLE_FORCE_COLOR=true ANSIBLE_HOST_KEY_CHECKING=false ANSIBLE_
↪SSH_ARGS='-o UserKnownHostsFile=/dev/null -o IdentitiesOnly=yes -o ControlMaster=auto -
↪o ControlPersist=60s' ansible-playbook --connection=ssh --timeout=30 --limit="default"␣
↪--inventory-file=/home/someone/coding-in-a-project/.vagrant/provisioners/ansible/
↪inventory -v playbook.yml
```

This information can be quite useful to debug integration issues and can also be used to manually execute Ansible from a shell, as explained in the next section.

Running Ansible Manually

Sometimes you may want to run Ansible manually against the machines. This is faster than kicking `vagrant provision` and pretty easy to do.

With our `Vagrantfile` example, Vagrant automatically creates an Ansible inventory file in `.vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory`. This inventory is configured according to the SSH tunnel that Vagrant automatically creates. A typical automatically-created inventory file for a single machine environment may look something like this:

```
# Generated by Vagrant

default ansible_host=127.0.0.1 ansible_port=2222 ansible_user='vagrant' ansible_ssh_
↪private_key_file='/home/someone/coding-in-a-project/.vagrant/machines/default/
↪virtualbox/private_key'
```

If you want to run Ansible manually, you will want to make sure to pass `ansible` or `ansible-playbook` commands the correct arguments, at least for the *inventory*.

```
$ ansible-playbook -i .vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory␣
↪playbook.yml
```

Advanced Usages

The "Tips and Tricks" chapter of the Ansible Provisioner documentation provides detailed information about more advanced Ansible features like:

- how to execute a playbook in parallel within a multi-machine environment
- how to integrate a local `ansible.cfg` configuration file

See also:

Vagrant Home
The Vagrant homepage with downloads

Vagrant Documentation
Vagrant Documentation

Ansible Provisioner
The Vagrant documentation for the Ansible provisioner

Vagrant Issue Tracker
The open issues for the Ansible provisioner in the Vagrant project

Working with playbooks
An introduction to playbooks

1.20.3 VMware REST Scenarios

The content on this page has moved to ansible_collections.vmware.vmware_rest.docsite.guide_vmware_rest.

1.21 Network Getting Started

Ansible collections support a wide range of vendors, device types, and actions, so you can manage your entire network with a single automation tool. With Ansible, you can:

- Automate repetitive tasks to speed routine network changes and free up your time for more strategic work
- Leverage the same simple, powerful, and agentless automation tool for network tasks that operations and development use
- Separate the data model (in a playbook or role) from the execution layer (through Ansible modules) to manage heterogeneous network devices
- Benefit from community and vendor-generated sample playbooks and roles to help accelerate network automation projects
- Communicate securely with network hardware over SSH or HTTPS

Who should use this guide?

This guide is intended for network engineers using Ansible for the first time. If you understand networks but have never used Ansible, work through the guide from start to finish.

This guide is also useful for experienced Ansible users automating network tasks for the first time. You can use Ansible commands, playbooks and modules to configure hubs, switches, routers, bridges and other network devices. But network modules are different from Linux/Unix and Windows modules, and you must understand some network-specific concepts to succeed. If you understand Ansible but have never automated a network task, start with the second section.

This guide introduces basic Ansible concepts and guides you through your first Ansible commands, playbooks and inventory entries.

1.21.1 Basic Concepts

These concepts are common to all uses of Ansible, including network automation. You need to understand them to use Ansible for network automation. This basic introduction provides the background you need to follow the examples in this guide.

Control node

The machine from which you run the Ansible CLI tools (`ansible-playbook` , `ansible`, `ansible-vault` and others). You can use any computer that meets the software requirements as a control node - laptops, shared desktops, and servers can all run Ansible. Multiple control nodes are possible, but Ansible itself does not coordinate across them, see `AAP` for such features.

Managed nodes

Also referred to as 'hosts', these are the target devices (servers, network appliances or any computer) you aim to manage with Ansible. Ansible is not normally installed on managed nodes, unless you are using `ansible-pull`, but this is rare and not the recommended setup.

Inventory

A list of managed nodes provided by one or more 'inventory sources'. Your inventory can specify information specific to each node, like IP address. It is also used for assigning groups, that both allow for node selection in the Play and bulk variable assignment. To learn more about inventory, see *the Working with Inventory* section. Sometimes an inventory source file is also referred to as a 'hostfile'.

Playbooks

They contain Plays (which are the basic unit of Ansible execution). This is both an 'execution concept' and how we describe the files on which `ansible-playbook` operates. Playbooks are written in YAML and are easy to read, write, share and understand. To learn more about playbooks, see *Ansible playbooks*.

Plays

The main context for Ansible execution, this playbook object maps managed nodes (hosts) to tasks. The Play contains variables, roles and an ordered lists of tasks and can be run repeatedly. It basically consists of an implicit loop over the mapped hosts and tasks and defines how to iterate over them.

Roles

A limited distribution of reusable Ansible content (tasks, handlers, variables, plugins, templates and files) for use inside of a Play. To use any Role resource, the Role itself must be imported into the Play.

Tasks

The definition of an 'action' to be applied to the managed host. Tasks must always be contained in a Play, directly or indirectly (Role, or imported/included task list file). You can execute a single task once with an ad hoc command using `ansible` or `ansible-console` (both create a virtual Play).

Handlers

A special form of a Task, that only executes when notified by a previous task which resulted in a 'changed' status.

Modules

The code or binaries that Ansible copies to and executes on each managed node (when needed) to accomplish the action defined in each Task. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. You can invoke a single module with a task, or invoke several different modules in a playbook. Ansible modules are grouped in collections. For an idea of how many collections Ansible includes, see the Collection Index.

Plugins

Pieces of code that expand Ansible's core capabilities, they can control how you connect to a managed node (connection plugins), manipulate data (filter plugins) and even control what is displayed in the console (callback plugins). See *Working with plugins* for details.

Collections

A format in which Ansible content is distributed that can contain playbooks, roles, modules, and plugins. You can install and use collections through Ansible Galaxy. To learn more about collections, see *Using Ansible collections*. Collection resources can be used independently and discretely from each other.

AAP

Short for 'Ansible Automation Platform'. This is a product that includes enterprise level features and integrates many tools of the Ansible ecosystem: ansible-core, awx, galaxyNG, and so on.

1.21.2 How Network Automation is Different

Network automation uses the basic Ansible concepts, but there are important differences in how the network modules work. This introduction prepares you to understand the exercises in this guide.

Execution on the control node

Unlike most Ansible modules, network modules do not run on the managed nodes. From a user's point of view, network modules work like any other modules. They work with ad hoc commands, playbooks, and roles. Behind the scenes, however, network modules use a different methodology than the other (Linux/Unix and Windows) modules use. Ansible is written and executed in Python. Because the majority of network devices can not run Python, the Ansible network modules are executed on the Ansible control node, where `ansible` or `ansible-playbook` runs.

Network modules also use the control node as a destination for backup files, for those modules that offer a `backup` option. With Linux/Unix modules, where a configuration file already exists on the managed node(s), the backup file gets written by default in the same directory as the new, changed file. Network modules do not update configuration files on the managed nodes, because network configuration is not written in files. Network modules write backup files on the control node, usually in the *backup* directory under the playbook root directory.

Multiple communication protocols

Because network modules execute on the control node instead of on the managed nodes, they can support multiple communication protocols. The communication protocol (XML over SSH, CLI over SSH, API over HTTPS) selected for each network module depends on the platform and the purpose of the module. Some network modules support only one protocol; some offer a choice. The most common protocol is CLI over SSH. You set the communication protocol with the `ansible_connection` variable:

Value of ansible_connection	Protocol	Requires	Persistent?
ansible.netcommon.network_cli	CLI over SSH	network_os setting	yes
ansible.netcommon.netconf	XML over SSH	network_os setting	yes
ansible.netcommon.httpapi	API over HTTP/HTTPS	network_os setting	yes
local	depends on provider	provider setting	no

Note: `ansible.netcommon.httpapi` deprecates `eos_eapi` and `nxos_nxapi`. See *Httpapi plugins* for details and an example.

The `ansible_connection: local` has been deprecated. Please use one of the persistent connection types listed above instead. With persistent connections, you can define the hosts and credentials only once, rather than in every task. You also need to set the `network_os` variable for the specific network platform you are communicating with. For more details on using each connection type on various platforms, see the *platform-specific* pages.

Collections organized by network platform

A network platform is a set of network devices with a common operating system that can be managed by an Ansible collection, for example:

- Arista: arista.eos
- Cisco: cisco.ios, cisco.iosxr, cisco.nxos
- Juniper: junipernetworks.junos
- VyOS vyos.vyos

All modules within a network platform share certain requirements. Some network platforms have specific differences - see the *platform-specific* documentation for details.

Privilege Escalation: `enable` mode, `become`, and `authorize`

Several network platforms support privilege escalation, where certain tasks must be done by a privileged user. On network devices this is called the `enable` mode (the equivalent of `sudo` in *nix administration). Ansible network modules offer privilege escalation for those network devices that support it. For details of which platforms support `enable` mode, with examples of how to use it, see the *platform-specific* documentation.

Using `become` for privilege escalation

Use the top-level Ansible parameter `become: true` with `become_method: enable` to run a task, play, or playbook with escalated privileges on any network platform that supports privilege escalation. You must use either `connection: network_cli` or `connection: httpapi` with `become: true` with `become_method: enable`. If you are using `network_cli` to connect Ansible to your network devices, a `group_vars` file would look like:

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.ios.ios
ansible_become: true
ansible_become_method: enable
```

For more information, see *Become and Networks*

1.21.3 Run Your First Command and Playbook

Put the concepts you learned to work with this quick tutorial. Install Ansible, execute a network configuration command manually, execute the same command with Ansible, then create a playbook so you can execute the command any time on multiple network devices.

Prerequisites

Before you work through this tutorial you need:

- Ansible 2.10 (or higher) installed
- One or more network devices that are compatible with Ansible
- Basic Linux command line knowledge
- Basic knowledge of network switch & router configuration

Install Ansible

Install Ansible using your preferred method. See *Installing Ansible*. Then return to this tutorial.

Confirm the version of Ansible (must be >= 2.10):

```
ansible --version
```

Establish a manual connection to a managed node

To confirm your credentials, connect to a network device manually and retrieve its configuration. Replace the sample user and device name with your real credentials. For example, for a VyOS router:

```
ssh my_vyos_user@vyos.example.net
show config
exit
```

This manual connection also establishes the authenticity of the network device, adding its RSA key fingerprint to your list of known hosts. (If you have connected to the device before, you have already established its authenticity.)

Run your first network Ansible command

Instead of manually connecting and running a command on the network device, you can retrieve its configuration with a single, stripped-down Ansible command:

```
ansible all -i vyos.example.net, -c ansible.netcommon.network_cli -u my_vyos_user -k -m␣
↪vyos.vyos.vyos_facts -e ansible_network_os=vyos.vyos.vyos
```

The flags in this command set seven values:

- the host group(s) to which the command should apply (in this case, all)
- the inventory (-i, the device or devices to target - without the trailing comma -i points to an inventory file)
- the connection method (-c, the method for connecting and executing ansible)
- the user (-u, the username for the SSH connection)
- the SSH connection method (-k, please prompt for the password)
- the module (-m, the Ansible module to run, using the fully qualified collection name (FQCN))
- an extra variable (-e, in this case, setting the network OS value)

NOTE: If you use `ssh-agent` with ssh keys, Ansible loads them automatically. You can omit `-k` flag.

Note: If you are running Ansible in a virtual environment, you will also need to add the variable `ansible_python_interpreter=/path/to/venv/bin/python`

Create and run your first network Ansible Playbook

If you want to run this command every day, you can save it in a playbook and run it with `ansible-playbook` instead of `ansible`. The playbook can store a lot of the parameters you provided with flags at the command line, leaving less to type at the command line. You need two files for this - a playbook and an inventory file.

1. Download `first_playbook.yml`, which looks like this:

```
---

- name: Network Getting Started First Playbook
  connection: ansible.netcommon.network_cli
  gather_facts: false
  hosts: all
  tasks:

    - name: Get config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
↪version }}"
```

The playbook sets three of the seven values from the command line above: the group (`hosts: all`), the connection method (`connection: ansible.netcommon.network_cli`) and the module (in each task). With those values set in the playbook, you can omit them on the command line. The playbook also adds a second task to show the config output. When a module runs in a playbook, the output is held in memory for use by future tasks instead of written to the console. The debug task here lets you see the results in your shell.

2. Run the playbook with the command:

```
ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos␣
↪first_playbook.yml
```

The playbook contains one play with two tasks, and should generate output like this:

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.
↪vyos first_playbook.yml

PLAY [Network Getting Started First Playbook]
****************************************************************************************

TASK [Get config for VyOS devices]
****************************************************************************************
ok: [vyos.example.net]

TASK [Display the config]
****************************************************************************************
ok: [vyos.example.net] => {
    "msg": "The hostname is vyos and the OS is VyOS 1.1.8"
}
```

3. Now that you can retrieve the device config, try updating it with Ansible. Download `first_playbook_ext.yml`, which is an extended version of the first playbook:

```
---

- name: Network Getting Started First Playbook Extended
  connection: ansible.netcommon.network_cli
  gather_facts: false
  hosts: all
  tasks:

    - name: Get config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
↪version }}"

    - name: Update the hostname
      vyos.vyos.vyos_config:
        backup: yes
        lines:
          - set system host-name vyos-changed

    - name: Get changed config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the changed config
      debug:
        msg: "The new hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_
↪net_version }}"
```

The extended first playbook has five tasks in a single play. Run it with the same command you used above. The output shows you the change Ansible made to the config:

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.
↪vyos first_playbook_ext.yml

PLAY [Network Getting Started First Playbook Extended]
*****************************************************************************************

TASK [Get config for VyOS devices]
*****************************************************************************************
ok: [vyos.example.net]

TASK [Display the config]
*****************************************************************************************
ok: [vyos.example.net] => {
    "msg": "The hostname is vyos and the OS is VyOS 1.1.8"
}

TASK [Update the hostname]
*****************************************************************************************
```

(continues on next page)

(continued from previous page)

```
changed: [vyos.example.net]

TASK [Get changed config for VyOS devices]
*******************************************************************************************************
ok: [vyos.example.net]

TASK [Display the changed config]
*******************************************************************************************************
ok: [vyos.example.net] => {
    "msg": "The new hostname is vyos-changed and the OS is VyOS 1.1.8"
}

PLAY RECAP
*******************************************************************************************************
vyos.example.net           : ok=5    changed=1    unreachable=0    failed=0
```

Gathering facts from network devices

The `gather_facts` keyword now supports gathering network device facts in standardized key/value pairs. You can feed these network facts into further tasks to manage the network device.

You can also use the new `gather_network_resources` parameter with the network `*_facts` modules (such as arista.eos.eos_facts) to return just a subset of the device configuration, as shown below.

```
- hosts: arista
  gather_facts: True
  gather_subset: interfaces
  module_defaults:
    arista.eos.eos_facts:
      gather_network_resources: interfaces
```

The playbook returns the following interface facts:

```
"network_resources": {
      "interfaces": [
          {
              "description": "test-interface",
              "enabled": true,
              "mtu": "512",
              "name": "Ethernet1"
          },
          {
              "enabled": true,
              "mtu": "3000",
              "name": "Ethernet2"
          },
          {
              "enabled": true,
              "name": "Ethernet3"
          },
          {
              "enabled": true,
```

(continues on next page)

(continued from previous page)

```
                "name": "Ethernet4"
            },
            {
                "enabled": true,
                "name": "Ethernet5"
            },
            {
                "enabled": true,
                "name": "Ethernet6"
            },
        ]
    }
```

Note that this returns a subset of what is returned by just setting `gather_subset:  interfaces`.

You can store these facts and use them directly in another task, such as with the eos_interfaces resource module.

1.21.4 Build Your Inventory

Running a playbook without an inventory requires several command-line flags. Also, running a playbook against a single device is not a huge efficiency gain over making the same change manually. The next step to harnessing the full power of Ansible is to use an inventory file to organize your managed nodes into groups with information like the `ansible_network_os` and the SSH user. A fully-featured inventory file can serve as the source of truth for your network. Using an inventory file, a single playbook can maintain hundreds of network devices with a single command. This page shows you how to build an inventory file, step by step.

Basic inventory

First, group your inventory logically. Best practice is to group servers and network devices by their What (application, stack or microservice), Where (datacenter or region), and When (development stage):

- **What**: db, web, leaf, spine
- **Where**: east, west, floor_19, building_A
- **When**: dev, test, staging, prod

Avoid spaces, hyphens, and preceding numbers (use `floor_19`, not `19th_floor`) in your group names. Group names are case sensitive.

This tiny example data center illustrates a basic group structure. You can group groups using the syntax `[metagroupname:children]` and listing groups as members of the metagroup. Here, the group `network` includes all leafs and all spines; the group `datacenter` includes all network devices plus all webservers.

```yaml
---

leafs:
  hosts:
    leaf01:
      ansible_host: 10.16.10.11
    leaf02:
      ansible_host: 10.16.10.12

spines:
  hosts:
    spine01:
      ansible_host: 10.16.10.13
    spine02:
      ansible_host: 10.16.10.14

network:
  children:
    leafs:
    spines:

webservers:
  hosts:
    webserver01:
      ansible_host: 10.16.10.15
    webserver02:
      ansible_host: 10.16.10.16

datacenter:
  children:
    network:
    webservers:
```

You can also create this same inventory in INI format.

```ini
[leafs]
leaf01
leaf02

[spines]
spine01
spine02

[network:children]
leafs
spines

[webservers]
webserver01
webserver02
```

(continues on next page)

(continued from previous page)

```
[datacenter:children]
network
webservers
```

Add variables to the inventory

Next, you can set values for many of the variables you needed in your first Ansible command in the inventory, so you can skip them in the `ansible-playbook` command. In this example, the inventory includes each network device's IP, OS, and SSH user. If your network devices are only accessible by IP, you must add the IP to the inventory file. If you access your network devices using hostnames, the IP is not necessary.

```
---

leafs:
  hosts:
    leaf01:
      ansible_host: 10.16.10.11
      ansible_network_os: vyos.vyos.vyos
      ansible_user: my_vyos_user
    leaf02:
      ansible_host: 10.16.10.12
      ansible_network_os: vyos.vyos.vyos
      ansible_user: my_vyos_user

spines:
  hosts:
    spine01:
      ansible_host: 10.16.10.13
      ansible_network_os: vyos.vyos.vyos
      ansible_user: my_vyos_user
    spine02:
      ansible_host: 10.16.10.14
      ansible_network_os: vyos.vyos.vyos
      ansible_user: my_vyos_user

network:
  children:
    leafs:
    spines:

webservers:
  hosts:
    webserver01:
      ansible_host: 10.16.10.15
      ansible_user: my_server_user
    webserver02:
      ansible_host: 10.16.10.16
      ansible_user: my_server_user

datacenter:
```

(continues on next page)

(continued from previous page)

```
  children:
    network:
    webservers:
```

Group variables within inventory

When devices in a group share the same variable values, such as OS or SSH user, you can reduce duplication and simplify maintenance by consolidating these into group variables:

```
---

leafs:
  hosts:
    leaf01:
      ansible_host: 10.16.10.11
    leaf02:
      ansible_host: 10.16.10.12
  vars:
    ansible_network_os: vyos.vyos.vyos
    ansible_user: my_vyos_user

spines:
  hosts:
    spine01:
      ansible_host: 10.16.10.13
    spine02:
      ansible_host: 10.16.10.14
  vars:
    ansible_network_os: vyos.vyos.vyos
    ansible_user: my_vyos_user

network:
  children:
    leafs:
    spines:

webservers:
  hosts:
    webserver01:
      ansible_host: 10.16.10.15
    webserver02:
      ansible_host: 10.16.10.16
  vars:
    ansible_user: my_server_user

datacenter:
  children:
    network:
    webservers:
```

Variable syntax

The syntax for variable values is different in inventory, in playbooks, and in the group_vars files, which are covered below. Even though playbook and group_vars files are both written in YAML, you use variables differently in each.

- In an ini-style inventory file you **must** use the syntax key=value for variable values: ansible_network_os=vyos.vyos.vyos.
- In any file with the .yml or .yaml extension, including playbooks and group_vars files, you **must** use YAML syntax: key: value.
- In group_vars files, use the full key name: ansible_network_os: vyos.vyos.vyos.
- In playbooks, use the short-form key name, which drops the ansible prefix: network_os: vyos.vyos.vyos.

Group inventory by platform

As your inventory grows, you may want to group devices by platform. This allows you to specify platform-specific variables easily for all devices on that platform:

```
---

leafs:
  hosts:
    leaf01:
      ansible_host: 10.16.10.11
    leaf02:
      ansible_host: 10.16.10.12

spines:
  hosts:
    spine01:
      ansible_host: 10.16.10.13
    spine02:
      ansible_host: 10.16.10.14

network:
  children:
    leafs:
    spines:
  vars:
    ansible_connection: ansible.netcommon.network_cli
    ansible_network_os: vyos.vyos.vyos
    ansible_user: my_vyos_user

webservers:
  hosts:
    webserver01:
      ansible_host: 10.16.10.15
    webserver02:
      ansible_host: 10.16.10.16
  vars:
    ansible_user: my_server_user

```

(continues on next page)

(continued from previous page)

```
datacenter:
  children:
    network:
    webservers:
```

With this setup, you can run `first_playbook.yml` with only two flags:

```
ansible-playbook -i inventory.yml -k first_playbook.yml
```

With the `-k` flag, you provide the SSH password(s) at the prompt. Alternatively, you can store SSH and other secrets and passwords securely in your group_vars files with `ansible-vault`. See *Protecting sensitive variables with ansible-vault* for details.

Verifying the inventory

You can use the *ansible-inventory* CLI command to display the inventory as Ansible sees it.

```
$ ansible-inventory -i test.yml --list
  {
    "_meta": {
        "hostvars": {
            "leaf01": {
                "ansible_connection": "ansible.netcommon.network_cli",
                "ansible_host": "10.16.10.11",
                "ansible_network_os": "vyos.vyos.vyos",
                "ansible_user": "my_vyos_user"
            },
            "leaf02": {
                "ansible_connection": "ansible.netcommon.network_cli",
                "ansible_host": "10.16.10.12",
                "ansible_network_os": "vyos.vyos.vyos",
                "ansible_user": "my_vyos_user"
            },
            "spine01": {
                "ansible_connection": "ansible.netcommon.network_cli",
                "ansible_host": "10.16.10.13",
                "ansible_network_os": "vyos.vyos.vyos",
                "ansible_user": "my_vyos_user"
            },
            "spine02": {
                "ansible_connection": "ansible.netcommon.network_cli",
                "ansible_host": "10.16.10.14",
                "ansible_network_os": "vyos.vyos.vyos",
                "ansible_user": "my_vyos_user"
            },
            "webserver01": {
                "ansible_host": "10.16.10.15",
                "ansible_user": "my_server_user"
            },
            "webserver02": {
                "ansible_host": "10.16.10.16",
                "ansible_user": "my_server_user"
```

(continues on next page)

(continued from previous page)

```
            }
        }
    },
    "all": {
        "children": [
            "datacenter",
            "ungrouped"
        ]
    },
    "datacenter": {
        "children": [
            "network",
            "webservers"
        ]
    },
    "leafs": {
        "hosts": [
            "leaf01",
            "leaf02"
        ]
    },
    "network": {
        "children": [
            "leafs",
            "spines"
        ]
    },
    "spines": {
        "hosts": [
            "spine01",
            "spine02"
        ]
    },
    "webservers": {
        "hosts": [
            "webserver01",
            "webserver02"
        ]
    }
}
```

Protecting sensitive variables with `ansible-vault`

The `ansible-vault` command provides encryption for files and/or individual variables like passwords. This tutorial will show you how to encrypt a single SSH password. You can use the commands below to encrypt other sensitive information, such as database passwords, privilege-escalation passwords and more.

First you must create a password for ansible-vault itself. It is used as the encryption key, and with this you can encrypt dozens of different passwords across your Ansible project. You can access all those secrets (encrypted values) with a single password (the ansible-vault password) when you run your playbooks. Here's a simple example.

1. Create a file and write your password for ansible-vault to it:

```
echo "my-ansible-vault-pw" > ~/my-ansible-vault-pw-file
```

2. Create the encrypted ssh password for your VyOS network devices, pulling your ansible-vault password from the file you just created:

```
ansible-vault encrypt_string --vault-id my_user@~/my-ansible-vault-pw-file 'VyOS_SSH_
↪password' --name 'ansible_password'
```

If you prefer to type your ansible-vault password rather than store it in a file, you can request a prompt:

```
ansible-vault encrypt_string --vault-id my_user@prompt 'VyOS_SSH_password' --name
↪'ansible_password'
```

and type in the vault password for `my_user`.

The *--vault-id* flag allows different vault passwords for different users or different levels of access. The output includes the user name `my_user` from your `ansible-vault` command and uses the YAML syntax `key:  value`:

```
ansible_password: !vault |
       $ANSIBLE_VAULT;1.2;AES256;my_user
       66386134653765386232383236303063623663343437643766386435663632343266393064373933
       3661666132363339303639353538316662616638356631650a316338316663666439383138353032
       63393934343937373637306162366265383461316334383132626462656463363630613832313562
       3837646266663835640a313164343535316666653031353763613037656362613535633538386539
       65656439626166666363323435613131643066353762333232326232323565376635
Encryption successful
```

This is an example using an extract from a YAML inventory, as the INI format does not support inline vaults:

```
...

vyos: # this is a group in yaml inventory, but you can also do under a host
  vars:
    ansible_connection: ansible.netcommon.network_cli
    ansible_network_os: vyos.vyos.vyos
    ansible_user: my_vyos_user
    ansible_password:  !vault |
         $ANSIBLE_VAULT;1.2;AES256;my_user
         66386134653765386232383236303063623663343437643766386435663632343266393064373933
         3661666132363339303639353538316662616638356631650a316338316663666439383138353032
         63393934343937373637306162366265383461316334383132626462656463363630613832313562
         3837646266663835640a313164343535316666653031353763613037656362613535633538386539
         65656439626166666363323435613131643066353762333232326232323565376635

 ...
```

To use an inline vaulted variables with an INI inventory you need to store it in a 'vars' file in YAML format, it can reside in host_vars/ or group_vars/ to be automatically picked up or referenced from a play through `vars_files` or `include_vars`.

To run a playbook with this setup, drop the `-k` flag and add a flag for your `vault-id`:

```
ansible-playbook -i inventory --vault-id my_user@~/my-ansible-vault-pw-file first_
↪playbook.yml
```

Or with a prompt instead of the vault password file:

```
ansible-playbook -i inventory --vault-id my_user@prompt first_playbook.yml
```

To see the original value, you can use the debug module. Please note if your YAML file defines the *ansible_connection* variable (as we used in our example), it will take effect when you execute the command below. To prevent this, please make a copy of the file without the ansible_connection variable.

```
cat vyos.yml | grep -v ansible_connection >> vyos_no_connection.yml

ansible localhost -m debug -a var="ansible_password" -e "@vyos_no_connection.yml" --ask-
↪vault-pass
Vault password:

localhost | SUCCESS => {
    "ansible_password": "VyOS_SSH_password"
}
```

Warning: Vault content can only be decrypted with the password that was used to encrypt it. If you want to stop using one password and move to a new one, you can update and re-encrypt existing vault content with `ansible-vault rekey myfile`, then provide the old password and the new password. Copies of vault content still encrypted with the old password can still be decrypted with old password.

For more details on building inventory files, see *the introduction to inventory*; for more details on ansible-vault, see *the full Ansible Vault documentation*.

Now that you understand the basics of commands, playbooks, and inventory, it's time to explore some more complex Ansible Network examples.

1.21.5 Use Ansible network roles

Roles are sets of Ansible defaults, files, tasks, templates, variables, and other Ansible components that work together. As you saw on *Run Your First Command and Playbook*, moving from a command to a playbook makes it easy to run multiple tasks and repeat the same tasks in the same order. Moving from a playbook to a role makes it even easier to reuse and share your ordered tasks. You can look at *Ansible Galaxy*, which lets you share your roles and use others' roles, either directly or as inspiration.

Understanding roles

So what exactly is a role, and why should you care? Ansible roles are basically playbooks broken up into a known file structure. Moving to roles from a playbook makes sharing, reading, and updating your Ansible workflow easier. Users can write their own roles. So for example, you don't have to write your own DNS playbook. Instead, you specify a DNS server and a role to configure it for you.

To simplify your workflow even further, the Ansible Network team has written a series of roles for common network use cases. Using these roles means you don't have to reinvent the wheel. Instead of writing and maintaining your own `create_vlan` playbooks or roles, you can concentrate on designing, codifying and maintaining the parser templates that describe your network topologies and inventory, and let Ansible's network roles do the work. See the network-related roles on Ansible Galaxy.

A sample DNS playbook

To demonstrate the concept of what a role is, the example `playbook.yml` below is a single YAML file containing a two-task playbook. This Ansible Playbook configures the hostname on a Cisco IOS XE device, then it configures the DNS (domain name system) servers.

```
---
- name: configure cisco routers
  hosts: routers
  connection: ansible.netcommon.network_cli
  gather_facts: no
  vars:
    dns: "8.8.8.8 8.8.4.4"

  tasks:
   - name: configure hostname
     cisco.ios.ios_config:
       lines: hostname {{ inventory_hostname }}

   - name: configure DNS
     cisco.ios.ios_config:
       lines: ip name-server {{dns}}
```

If you run this playbook using the `ansible-playbook` command, you'll see the output below. This example used `-l` option to limit the playbook to only executing on the **rtr1** node.

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr1

PLAY [configure cisco routers] *************************************************

TASK [configure hostname] ******************************************************
changed: [rtr1]

TASK [configure DNS] ***********************************************************
changed: [rtr1]

PLAY RECAP *********************************************************************
rtr1                       : ok=2    changed=2    unreachable=0    failed=0
```

This playbook configured the hostname and DNS servers. You can verify that configuration on the Cisco IOS XE **rtr1** router:

```
rtr1#sh run | i name
hostname rtr1
ip name-server 8.8.8.8 8.8.4.4
```

Convert the playbook into a role

The next step is to convert this playbook into a reusable role. You can create the directory structure manually, or you can use `ansible-galaxy init` to create the standard framework for a role.

```
[user@ansible ~]$ ansible-galaxy init system-demo
[user@ansible ~]$ cd system-demo/
[user@ansible system-demo]$ tree
.
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
  └── main.yml
```

This first demonstration uses only the **tasks** and **vars** directories. The directory structure would look as follows:

```
[user@ansible system-demo]$ tree
.
├── tasks
│   └── main.yml
└── vars
    └── main.yml
```

Next, move the content of the `vars` and `tasks` sections from the original Ansible Playbook into the role. First, move the two tasks into the `tasks/main.yml` file:

```
[user@ansible system-demo]$ cat tasks/main.yml
---
- name: configure hostname
  cisco.ios.ios_config:
    lines: hostname {{ inventory_hostname }}

- name: configure DNS
  cisco.ios.ios_config:
    lines: ip name-server {{dns}}
```

Next, move the variables into the `vars/main.yml` file:

```
[user@ansible system-demo]$ cat vars/main.yml
---
dns: "8.8.8.8 8.8.4.4"
```

Finally, modify the original Ansible Playbook to remove the `tasks` and `vars` sections and add the keyword `roles` with the name of the role, in this case `system-demo`. You'll have this playbook:

```
---
- name: configure cisco routers
  hosts: routers
  connection: ansible.netcommon.network_cli
  gather_facts: no

  roles:
    - system-demo
```

To summarize, this demonstration now has a total of three directories and three YAML files. There is the `system-demo` folder, which represents the role. This `system-demo` contains two folders, `tasks` and `vars`. There is a `main.yml` is each respective folder. The `vars/main.yml` contains the variables from `playbook.yml`. The `tasks/main.yml` contains the tasks from `playbook.yml`. The `playbook.yml` file has been modified to call the role rather than specifying vars and tasks directly. Here is a tree of the current working directory:

```
[user@ansible ~]$ tree
.
├── playbook.yml
└── system-demo
    ├── tasks
    │   └── main.yml
    └── vars
        └── main.yml
```

Running the playbook results in identical behavior with slightly different output:

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr1

PLAY [configure cisco routers] *************************************************

TASK [system-demo : configure hostname] ****************************************
ok: [rtr1]

TASK [system-demo : configure DNS] *********************************************
ok: [rtr1]

PLAY RECAP *********************************************************************
rtr1             : ok=2    changed=0    unreachable=0    failed=0
```

As seen above each task is now prepended with the role name, in this case `system-demo`. When running a playbook that contains several roles, this will help pinpoint where a task is being called from. This playbook returned `ok` instead of `changed` because it has identical behavior for the single file playbook we started from.

As before, the playbook will generate the following configuration on a Cisco IOS-XE router:

```
rtr1#sh run | i name
hostname rtr1
```

(continues on next page)

(continued from previous page)

```
ip name-server 8.8.8.8 8.8.4.4
```

This is why Ansible roles can be simply thought of as deconstructed playbooks. They are simple, effective and reusable. Now another user can simply include the `system-demo` role instead of having to create a custom "hard coded" playbook.

Variable precedence

What if you want to change the DNS servers? You aren't expected to change the `vars/main.yml` within the role structure. Ansible has many places where you can specify variables for a given play. See *Using Variables* for details on variables and precedence. There are actually 21 places to put variables. While this list can seem overwhelming at first glance, the vast majority of use cases only involve knowing the spot for variables of least precedence and how to pass variables with most precedence. See *Variable precedence: Where should I put a variable?* for more guidance on where you should put variables.

Lowest precedence

The lowest precedence is the `defaults` directory within a role. This means all the other 20 locations you could potentially specify the variable will all take higher precedence than `defaults`, no matter what. To immediately give the vars from the `system-demo` role the least precedence, rename the `vars` directory to `defaults`.

```
[user@ansible system-demo]$ mv vars defaults
[user@ansible system-demo]$ tree
.
├── defaults
│   └── main.yml
├── tasks
│   └── main.yml
```

Add a new `vars` section to the playbook to override the default behavior (where the variable `dns` is set to 8.8.8.8 and 8.8.4.4). For this demonstration, set `dns` to 1.1.1.1, so `playbook.yml` becomes:

```
---
- name: configure cisco routers
  hosts: routers
  connection: ansible.netcommon.network_cli
  gather_facts: no
  vars:
    dns: 1.1.1.1
  roles:
    - system-demo
```

Run this updated playbook on **rtr2**:

```
[user@ansible ~]$ ansible-playbook playbook.yml -l rtr2
```

The configuration on the **rtr2** Cisco router will look as follows:

```
rtr2#sh run | i name-server
ip name-server 1.1.1.1
```

The variable configured in the playbook now has precedence over the `defaults` directory. In fact, any other spot you configure variables would win over the values in the `defaults` directory.

Highest precedence

Specifying variables in the `defaults` directory within a role will always take the lowest precedence, while specifying `vars` as extra vars with the `-e` or `--extra-vars=` will always take the highest precedence, no matter what. Re-running the playbook with the `-e` option overrides both the `defaults` directory (8.8.4.4 and 8.8.8.8) as well as the newly created `vars` within the playbook that contains the 1.1.1.1 dns server.

```
[user@ansible ~]$ ansible-playbook playbook.yml -e "dns=192.168.1.1" -l rtr3
```

The result on the Cisco IOS XE router will only contain the highest precedence setting of 192.168.1.1:

```
rtr3#sh run | i name-server
ip name-server 192.168.1.1
```

How is this useful? Why should you care? Extra vars are commonly used by network operators to override defaults. A powerful example of this is with the Job Template Survey feature on AWX or the *Red Hat Ansible Automation Platform*. It is possible through the web UI to prompt a network operator to fill out parameters with a Web form. This can be really simple for non-technical playbook writers to execute a playbook using their Web browser.

Update an installed role

The Ansible Galaxy page for a role lists all available versions. To update a locally installed role to a new or different version, use the `ansible-galaxy install` command with the version and `--force` option. You may also need to manually update any dependent roles to support this version. See the role **Read Me** tab in Galaxy for dependent role minimum version requirements.

```
[user@ansible]$ ansible-galaxy install mynamespace.my_role,v2.7.1 --force
```

See also:

Ansible Galaxy documentation
Ansible Galaxy user guide

1.21.6 Beyond the basics

This page introduces some concepts that help you manage your Ansible workflow with directory structure and source control. Like the Basic Concepts at the beginning of this guide, these intermediate concepts are common to all uses of Ansible.

A typical Ansible filetree

Ansible expects to find certain files in certain places. As you expand your inventory and create and run more network playbooks, keep your files organized in your working Ansible project directory like this:

```
.
├── backup
│   ├── vyos.example.net_config.2018-02-08@11:10:15
│   ├── vyos.example.net_config.2018-02-12@08:22:41
├── first_playbook.yml
├── inventory
├── group_vars
│   ├── vyos.yml
│   └── eos.yml
├── roles
│   ├── static_route
│   └── system
├── second_playbook.yml
└── third_playbook.yml
```

The `backup` directory and the files in it get created when you run modules like `vyos_config` with the `backup: yes` parameter.

Tracking changes to inventory and playbooks: source control with git

As you expand your inventory, roles and playbooks, you should place your Ansible projects under source control. We recommend `git` for source control. `git` provides an audit trail, letting you track changes, roll back mistakes, view history and share the workload of managing, maintaining and expanding your Ansible ecosystem. There are plenty of tutorials and guides to using `git` available.

1.21.7 Working with network connection options

Network modules can support multiple connection protocols, such as `ansible.netcommon.network_cli`, `ansible.netcommon.netconf`, and `ansible.netcommon.httpapi`. These connections include some common options you can set to control how the connection to your network device behaves.

Common options are:

- `become` and `become_method` as described in *Privilege Escalation: enable mode, become, and authorize*.
- `network_os` - set to match your network platform you are communicating with. See the *platform-specific* pages.
- `remote_user` as described in *Setting a remote user*.
- Timeout options - `persistent_command_timeout`, `persistent_connect_timeout`, and `timeout`.

Setting timeout options

When communicating with a remote device, you have control over how long Ansible maintains the connection to that device, as well as how long Ansible waits for a command to complete on that device. Each of these options can be set as variables in your playbook files, environment variables, or settings in your *ansible.cfg file*.

For example, the three options for controlling the connection timeout are as follows.

Using vars (per task):

```
- name: save running-config
  cisco.ios.ios_command:
    commands: copy running-config startup-config
    vars:
      ansible_command_timeout: 30
```

Using the environment variable:

```
$export ANSIBLE_PERSISTENT_COMMAND_TIMEOUT=30
```

Using the global configuration (in `ansible.cfg`)

```
[persistent_connection]
command_timeout = 30
```

See *Variable precedence: Where should I put a variable?* for details on the relative precedence of each of these variables. See the individual connection type to understand each option.

1.21.8 Resources and next steps

Documents

Read more about Ansible for Network Automation:

- *Network Platform Options*
- Network Automation on the Ansible website
- Ansible Network Blog posts

Events (on video and in person)

All sessions at Ansible events are recorded and include many Network-related topics (use Filter by Category to view only Network topics). You can also join us for future events in your area. See:

- Recorded AnsibleFests
- Recorded AnsibleAutomates
- Upcoming Ansible Events page.

GitHub repos

Ansible hosts module code, examples, demonstrations, and other content on GitHub. Anyone with a GitHub account is able to create Pull Requests (PRs) or issues on these repos:

- Network-Automation is an open community for all things network automation. Have an idea, some playbooks, or roles to share? Email ansible-network@redhat.com and we will add you as a contributor to the repository.
- Ansible collections is the main repository for Ansible-maintained and community collections, including collections for network devices.

Chat channels

Got questions? Chat with us on:

- the `#ansible-network` channel (using Matrix at ansible.im or using IRC at irc.libera.chat)
- Ansible Network Slack - Network & Security Automation Slack community. Check out the #devel channel for discussions on module and plugin development.

1.22 Network Advanced Topics

Once you have mastered the basics of network automation with Ansible, as presented in *Network Getting Started*, use this guide understand platform-specific details, optimization, and troubleshooting tips for Ansible for network automation.

Who should use this guide?

This guide is intended for network engineers using Ansible for automation. It covers advanced topics. If you understand networks and Ansible, this guide is for you. You may read through the entire guide if you choose, or use the links below to find the specific information you need.

If you're new to Ansible, or new to using Ansible for network automation, start with the *Network Getting Started*.

1.22.1 Network Resource Modules

Ansible network resource modules simplify and standardize how you manage different network devices. Network devices separate configuration into sections (such as interfaces and VLANs) that apply to a network service. Ansible network resource modules take advantage of this to allow you to configure subsections or *resources* within the network device configuration. Network resource modules provide a consistent experience across different network devices.

Network resource module states

You use the network resource modules by assigning a state to what you want the module to do. The resource modules support the following states:

merged
  Ansible merges the on-device configuration with the provided configuration in the task.

replaced
  Ansible replaces the on-device configuration subsection with the provided configuration subsection in the task.

overridden
  Ansible overrides the on-device configuration for the resource with the provided configuration in the task. Use caution with this state as you could remove your access to the device (for example, by overriding the management interface configuration).

deleted
  Ansible deletes the on-device configuration subsection and restores any default settings.

gathered
  Ansible displays the resource details gathered from the network device and accessed with the `gathered` key in the result.

rendered
  Ansible renders the provided configuration in the task in the device-native format (for example, Cisco IOS CLI). Ansible returns this rendered configuration in the `rendered` key in the result. Note this state does not communicate with the network device and can be used offline.

parsed
  Ansible parses the configuration from the `running_config` option into Ansible structured data in the `parsed` key in the result. Note this does not gather the configuration from the network device so this state can be used offline.

Using network resource modules

This example configures the L3 interface resource on a Cisco IOS device, based on different state settings.

```
- name: configure l3 interface
  cisco.ios.ios_l3_interfaces:
    config: "{{ config }}"
    state: <state>
```

The following table shows an example of how an initial resource configuration changes with this task for different states.

Resource starting configuration	task-provided configuration (YAML)	Final resource configuration on device

`interface loopback100`
` ip address 10.10.1.100 ↪255.255.255.0`
` ipv6 address FC00:100/64`
	`config:`
	`- ipv6:`
	`  - address: fc00::100/64`
	`  - address: fc00::101/64`
	`  name: loopback100`
		merged
		`interface loopback100`
		` ip address 10.10.1. ↪100 255.255.255.0`
		` ipv6 address ↪FC00:100/64`
		` ipv6 address ↪FC00:101/64`
		
		replaced
		`interface loopback100`
		` no ip address`
		` ipv6 address ↪FC00:100/64`
		` ipv6 address ↪FC00:101/64`
		
		overridden
		Incorrect use case. This would remove all interfaces from the device
		(including the mgmt interface) except
		the configured loopback100
		
		deleted
		`interface loopback100`
		` no ip address`

Network resource modules return the following details:

- The *before* state - the existing resource configuration before the task was executed.
- The *after* state - the new resource configuration that exists on the network device after the task was executed.
- Commands - any commands configured on the device.

```
ok: [nxos101] =>
  result:
    after:
      contact: IT Support
      location: Room E, Building 6, Seattle, WA 98134
      users:
      - algorithm: md5
        group: network-admin
        localized_key: true
```

(continues on next page)

(continued from previous page)

```
        password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
        privacy_password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
        username: admin
    before:
      contact: IT Support
      location: Room E, Building 5, Seattle HQ
      users:
      - algorithm: md5
        group: network-admin
        localized_key: true
        password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
        privacy_password: '0x73fd9a2cc8c53ed3dd4ed8f4ff157e69'
        username: admin
    changed: true
    commands:
    - snmp-server location Room E, Building 6, Seattle, WA 98134
    failed: false
```

Example: Verifying the network device configuration has not changed

The following playbook uses the arista.eos.eos_l3_interfaces module to gather a subset of the network device configuration (Layer 3 interfaces only) and verifies the information is accurate and has not changed. This playbook passes the results of arista.eos.eos_facts directly to the `arista.eos.eos_l3_interfaces` module.

```
- name: Example of facts being pushed right back to device.
  hosts: arista
  gather_facts: false
  tasks:
    - name: grab arista eos facts
      arista.eos.eos_facts:
        gather_subset: min
        gather_network_resources: l3_interfaces

- name: Ensure that the IP address information is accurate.
  arista.eos.eos_l3_interfaces:
    config: "{{ ansible_network_resources['l3_interfaces'] }}"
    register: result

- name: Ensure config did not change.
  assert:
    that: not result.changed
```

Example: Acquiring and updating VLANs on a network device

This example shows how you can use resource modules to:

1. Retrieve the current configuration on a network device.
2. Save that configuration locally.
3. Update that configuration and apply it to the network device.

This example uses the `cisco.ios.ios_vlans` resource module to retrieve and update the VLANs on an IOS device.

1. Retrieve the current IOS VLAN configuration:

```
- name: Gather VLAN information as structured data
  cisco.ios.ios_facts:
     gather_subset:
      - '!all'
      - '!min'
     gather_network_resources:
     - 'vlans'
```

2. Store the VLAN configuration locally:

```
- name: Store VLAN facts to host_vars
  copy:
    content: "{{ ansible_network_resources | to_nice_yaml }}"
    dest: "{{ playbook_dir }}/host_vars/{{ inventory_hostname }}"
```

3. Modify the stored file to update the VLAN configuration locally.
4. Merge the updated VLAN configuration with the existing configuration on the device:

```
- name: Make VLAN config changes by updating stored facts on the controller.
  cisco.ios.ios_vlans:
    config: "{{ vlans }}"
    state: merged
  tags: update_config
```

See also:

Network Features in Ansible 2.9
A introductory blog post on network resource modules.

Deep Dive into Network Resource Modules
A deeper dive presentation into network resource modules.

1.22.2 Ansible Network Examples

This document describes some examples of using Ansible to manage your network infrastructure.

Prerequisites

This example requires the following:

- **Ansible 2.10** (or higher) installed. See *Installing Ansible* for more information.
- One or more network devices that are compatible with Ansible.
- Basic understanding of YAML *YAML Syntax*.
- Basic understanding of Jinja2 templates. See *Templating (Jinja2)* for more information.
- Basic Linux command line use.
- Basic knowledge of network switch & router configurations.

Groups and variables in an inventory file

An `inventory` file is a YAML or INI-like configuration file that defines the mapping of hosts into groups.

In our example, the inventory file defines the groups `eos`, `ios`, `vyos` and a "group of groups" called `switches`. Further details about subgroups and inventory files can be found in the *Ansible inventory Group documentation*.

Because Ansible is a flexible tool, there are a number of ways to specify connection information and credentials. We recommend using the `[my_group:vars]` capability in your inventory file.

```
[all:vars]
# these defaults can be overridden for any group in the [group:vars] section
ansible_connection=ansible.netcommon.network_cli
ansible_user=ansible

[switches:children]
```

(continues on next page)

(continued from previous page)

```
eos
ios
vyos

[eos]
veos01 ansible_host=veos-01.example.net
veos02 ansible_host=veos-02.example.net
veos03 ansible_host=veos-03.example.net
veos04 ansible_host=veos-04.example.net

[eos:vars]
ansible_become=yes
ansible_become_method=enable
ansible_network_os=arista.eos.eos
ansible_user=my_eos_user
ansible_password=my_eos_password

[ios]
ios01 ansible_host=ios-01.example.net
ios02 ansible_host=ios-02.example.net
ios03 ansible_host=ios-03.example.net

[ios:vars]
ansible_become=yes
ansible_become_method=enable
ansible_network_os=cisco.ios.ios
ansible_user=my_ios_user
ansible_password=my_ios_password

[vyos]
vyos01 ansible_host=vyos-01.example.net
vyos02 ansible_host=vyos-02.example.net
vyos03 ansible_host=vyos-03.example.net

[vyos:vars]
ansible_network_os=vyos.vyos.vyos
ansible_user=my_vyos_user
ansible_password=my_vyos_password
```

If you use ssh-agent, you do not need the `ansible_password` lines. If you use ssh keys, but not ssh-agent, and you have multiple keys, specify the key to use for each connection in the `[group:vars]` section with `ansible_ssh_private_key_file=/path/to/correct/key`. For more information on `ansible_ssh_` options see *Connecting to hosts: behavioral inventory parameters*.

Warning: Never store passwords in plain text.

Ansible vault for password encryption

The "Vault" feature of Ansible allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plain text in your playbooks or roles. These vault files can then be distributed or placed in source control. See *Using encrypted variables and files* for more information.

Here's what it would look like if you specified your SSH passwords (encrypted with Ansible Vault) among your variables:

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: vyos.vyos.vyos
ansible_user: my_vyos_user
ansible_ssh_pass: !vault |
                  $ANSIBLE_VAULT;1.1;AES256
                 ␣
↪39336231636137663964343966653162353431333566633762393034646462353062633264303765
                 ␣
↪6331643066663534383564343537343334633031656538370a333737656236393835383863306466
                 ␣
↪62633364653238323333633337313163616566383836643030336631333431623631396364663533
                 ␣
↪3665626431626532630a353564323566316162613432373738333064366130303637616239396438
                  9853
```

Common inventory variables

The following variables are common for all platforms in the inventory, though they can be overwritten for a particular inventory group or host.

ansible_connection
: Ansible uses the ansible-connection setting to determine how to connect to a remote device. When working with Ansible Networking, set this to an appropriate network connection option, such as``ansible.netcommon.network_cli``, so Ansible treats the remote node as a network device with a limited execution environment. Without this setting, Ansible would attempt to use ssh to connect to the remote and execute the Python script on the network device, which would fail because Python generally isn't available on network devices.

ansible_network_os
: Informs Ansible which Network platform this hosts corresponds to. This is required when using the `ansible.netcommon.*` connection options.

ansible_user
: The user to connect to the remote device (switch) as. Without this the user that is running `ansible-playbook` would be used. Specifies which user on the network device the connection

ansible_password
: The corresponding password for `ansible_user` to log in as. If not specified SSH key will be used.

ansible_become
: If enable mode (privilege mode) should be used, see the next section.

ansible_become_method
: Which type of *become* should be used, for `network_cli` the only valid choice is `enable`.

Privilege escalation

Certain network platforms, such as Arista EOS and Cisco IOS, have the concept of different privilege modes. Certain network modules, such as those that modify system state including users, will only work in high privilege states. Ansible supports `become` when using `connection: ansible.netcommon.network_cli`. This allows privileges to be raised for the specific tasks that need them. Adding `become: yes` and `become_method: enable` informs Ansible to go into privilege mode before executing the task, as shown here:

```
[eos:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=arista.eos.eos
ansible_become=yes
ansible_become_method=enable
```

For more information, see the *using become with network modules* guide.

Jump hosts

If the Ansible Controller does not have a direct route to the remote device and you need to use a Jump Host, please see the *Ansible Network Proxy Command* guide for details on how to achieve this.

Example 1: collecting facts and creating backup files with a playbook

Ansible facts modules gather system information 'facts' that are available to the rest of your playbook.

Ansible Networking ships with a number of network-specific facts modules. In this example, we use the `_facts` modules arista.eos.eos_facts, cisco.ios.ios_facts and vyos.vyos.vyos_facts to connect to the remote networking device. As the credentials are not explicitly passed with module arguments, Ansible uses the username and password from the inventory file.

Ansible's "Network Fact modules" gather information from the system and store the results in facts prefixed with `ansible_net_`. The data collected by these modules is documented in the *Return Values* section of the module docs, in this case arista.eos.eos_facts and vyos.vyos.vyos_facts. We can use the facts, such as `ansible_net_version` late on in the "Display some facts" task.

To ensure we call the correct mode (`*_facts`) the task is conditionally run based on the group defined in the inventory file, for more information on the use of conditionals in Ansible Playbooks see *Basic conditionals with when*.

In this example, we will create an inventory file containing some network switches, then run a playbook to connect to the network devices and return some information about them.

Step 1: Creating the inventory

First, create a file called `inventory`, containing:

```
[switches:children]
eos
ios
vyos

[eos]
eos01.example.net

```

(continues on next page)

(continued from previous page)

```
[ios]
ios01.example.net

[vyos]
vyos01.example.net
```

Step 2: Creating the playbook

Next, create a playbook file called `facts-demo.yml` containing the following:

```
- name: "Demonstrate connecting to switches"
  hosts: switches
  gather_facts: no

  tasks:
    ###
    # Collect data
    #
    - name: Gather facts (eos)
      arista.eos.eos_facts:
      when: ansible_network_os == 'arista.eos.eos'

    - name: Gather facts (ios)
      cisco.ios.ios_facts:
      when: ansible_network_os == 'cisco.ios.ios'

    - name: Gather facts (vyos)
      vyos.vyos.vyos_facts:
      when: ansible_network_os == 'vyos.vyos.vyos'

    ###
    # Demonstrate variables
    #
    - name: Display some facts
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_
↪version }}"

    - name: Facts from a specific host
      debug:
        var: hostvars['vyos01.example.net']

    - name: Write facts to disk using a template
      copy:
        content: |
          #jinja2: lstrip_blocks: True
          EOS device info:
            {% for host in groups['eos'] %}
            Hostname: {{ hostvars[host].ansible_net_hostname }}
            Version: {{ hostvars[host].ansible_net_version }}
            Model: {{ hostvars[host].ansible_net_model }}
```

(continues on next page)

(continued from previous page)

```
          Serial: {{ hostvars[host].ansible_net_serialnum }}
          {% endfor %}

        IOS device info:
          {% for host in groups['ios'] %}
          Hostname: {{ hostvars[host].ansible_net_hostname }}
          Version: {{ hostvars[host].ansible_net_version }}
          Model: {{ hostvars[host].ansible_net_model }}
          Serial: {{ hostvars[host].ansible_net_serialnum }}
          {% endfor %}

        VyOS device info:
          {% for host in groups['vyos'] %}
          Hostname: {{ hostvars[host].ansible_net_hostname }}
          Version: {{ hostvars[host].ansible_net_version }}
          Model: {{ hostvars[host].ansible_net_model }}
          Serial: {{ hostvars[host].ansible_net_serialnum }}
          {% endfor %}
      dest: /tmp/switch-facts
    run_once: yes

  ###
  # Get running configuration
  #

  - name: Backup switch (eos)
    arista.eos.eos_config:
      backup: yes
    register: backup_eos_location
    when: ansible_network_os == 'arista.eos.eos'

  - name: backup switch (vyos)
    vyos.vyos.vyos_config:
      backup: yes
    register: backup_vyos_location
    when: ansible_network_os == 'vyos.vyos.vyos'

  - name: Create backup dir
    file:
      path: "/tmp/backups/{{ inventory_hostname }}"
      state: directory
      recurse: yes

  - name: Copy backup files into /tmp/backups/ (eos)
    copy:
      src: "{{ backup_eos_location.backup_path }}"
      dest: "/tmp/backups/{{ inventory_hostname }}/{{ inventory_hostname }}.bck"
    when: ansible_network_os == 'arista.eos.eos'

  - name: Copy backup files into /tmp/backups/ (vyos)
    copy:
      src: "{{ backup_vyos_location.backup_path }}"
```

(continues on next page)

(continued from previous page)

```
        dest: "/tmp/backups/{{ inventory_hostname }}/{{ inventory_hostname }}.bck"
      when: ansible_network_os == 'vyos.vyos.vyos'
```

Step 3: Running the playbook

To run the playbook, run the following from a console prompt:

```
ansible-playbook -i inventory facts-demo.yml
```

This should return output similar to the following:

```
PLAY RECAP
eos01.example.net          : ok=7    changed=2    unreachable=0    failed=0
ios01.example.net          : ok=7    changed=2    unreachable=0    failed=0
vyos01.example.net         : ok=6    changed=2    unreachable=0    failed=0
```

Step 4: Examining the playbook results

Next, look at the contents of the file we created containing the switch facts:

```
cat /tmp/switch-facts
```

You can also look at the backup files:

```
find /tmp/backups
```

If *ansible-playbook* fails, please follow the debug steps in *Network Debug and Troubleshooting Guide*.

Example 2: simplifying playbooks with platform-independent modules

(This example originally appeared in the Deep Dive on cli_command for Network Automation blog post by Sean Cavanaugh -@IPvSean).

If you have two or more network platforms in your environment, you can use the platform-independent modules to simplify your playbooks. You can use platform-independent modules such as `ansible.netcommon.cli_command` or `ansible.netcommon.cli_config` in place of the platform-specific modules such as `arista.eos.eos_config`, `cisco.ios.ios_config`, and `junipernetworks.junos.junos_config`. This reduces the number of tasks and conditionals you need in your playbooks.

Note: Platform-independent modules require the ansible.netcommon.network_cli connection plugin.

Sample playbook with platform-specific modules

This example assumes three platforms, Arista EOS, Cisco NXOS, and Juniper JunOS. Without the platform-independent modules, a sample playbook might contain the following three tasks with platform-specific commands:

```
---
- name: Run Arista command
  arista.eos.eos_command:
    commands: show ip int br
  when: ansible_network_os == 'arista.eos.eos'

- name: Run Cisco NXOS command
  cisco.nxos.nxos_command:
    commands: show ip int br
  when: ansible_network_os == 'cisco.nxos.nxos'

- name: Run Vyos command
  vyos.vyos.vyos_command:
    commands: show interface
  when: ansible_network_os == 'vyos.vyos.vyos'
```

Simplified playbook with `cli_command` platform-independent module

You can replace these platform-specific modules with the platform-independent `ansible.netcommon.cli_command` module as follows:

```
---
- hosts: network
  gather_facts: false
  connection: ansible.netcommon.network_cli

  tasks:
    - name: Run cli_command on Arista and display results
      block:
      - name: Run cli_command on Arista
        ansible.netcommon.cli_command:
          command: show ip int br
        register: result

      - name: Display result to terminal window
        debug:
          var: result.stdout_lines
      when: ansible_network_os == 'arista.eos.eos'

    - name: Run cli_command on Cisco IOS and display results
      block:
      - name: Run cli_command on Cisco IOS
        ansible.netcommon.cli_command:
          command: show ip int br
        register: result

      - name: Display result to terminal window
```

(continues on next page)

(continued from previous page)

```
        debug:
          var: result.stdout_lines
      when: ansible_network_os == 'cisco.ios.ios'

    - name: Run cli_command on Vyos and display results
      block:
      - name: Run cli_command on Vyos
        ansible.netcommon.cli_command:
          command: show interfaces
        register: result

      - name: Display result to terminal window
        debug:
          var: result.stdout_lines
      when: ansible_network_os == 'vyos.vyos.vyos'
```

If you use groups and group_vars by platform type, this playbook can be further simplified to :

```
---
- name: Run command and print to terminal window
  hosts: routers
  gather_facts: false

  tasks:
    - name: Run show command
      ansible.netcommon.cli_command:
        command: "{{show_interfaces}}"
      register: command_output
```

You can see a full example of this using group_vars and also a configuration backup example at Platform-independent examples.

Using multiple prompts with the `ansible.netcommon.cli_command`

The `ansible.netcommon.cli_command` also supports multiple prompts.

```
---
- name: Change password to default
  ansible.netcommon.cli_command:
    command: "{{ item }}"
    prompt:
      - "New password"
      - "Retype new password"
    answer:
      - "mypassword123"
      - "mypassword123"
    check_all: True
  loop:
    - "configure"
    - "rollback"
    - "set system root-authentication plain-text-password"
    - "commit"
```

See the ansible.netcommon.cli_command for full documentation on this command.

Implementation Notes

Demo variables

Although these tasks are not needed to write data to disk, they are used in this example to demonstrate some methods of accessing facts about the given devices or a named host.

Ansible `hostvars` allows you to access variables from a named host. Without this we would return the details for the current host, rather than the named host.

For more information, see *Information about Ansible: magic variables*.

Get running configuration

The arista.eos.eos_config and vyos.vyos.vyos_config modules have a `backup:` option that when set will cause the module to create a full backup of the current `running-config` from the remote device before any changes are made. The backup file is written to the `backup` folder in the playbook root directory. If the directory does not exist, it is created.

To demonstrate how we can move the backup file to a different location, we register the result and move the file to the path stored in `backup_path`.

Note that when using variables from tasks in this way we use double quotes (`"`) and double curly-brackets (`{{...}}` to tell Ansible that this is a variable.

Troubleshooting

If you receive an connection error please double check the inventory and playbook for typos or missing lines. If the issue still occurs follow the debug steps in *Network Debug and Troubleshooting Guide*.

See also:

- network_guide
- *How to build your inventory*
- *Keeping vaulted variables visible*

1.22.3 Parsing semi-structured text with Ansible

The cli_parse module parses semi-structured data such as network configurations into structured data to allow programmatic use of the data from that device. You can pull information from a network device and update a CMDB in one playbook. Use cases include automated troubleshooting, creating dynamic documentation, updating IPAM (IP address management) tools and so on.

Understanding the CLI parser

The ansible.utils collection version 1.0.0 or later includes the cli_parse module that can run CLI commands and parse the semi-structured text output. You can use the `cli_parse` module on a device, host, or platform that only supports a command-line interface and the commands issued return semi-structured text. The `cli_parse` module can either run a CLI command on a device and return a parsed result or can simply parse any text document. The `cli_parse` module includes cli_parser plugins to interface with a variety of parsing engines.

Why parse the text?

Parsing semi-structured data such as network configurations into structured data allows programmatic use of the data from that device. Use cases include automated troubleshooting, creating dynamic documentation, updating IPAM (IP address management) tools and so on. You may prefer to do this with Ansible natively to take advantage of native Ansible constructs such as:

- The `when` clause to conditionally run other tasks or roles
- The `assert` module to check configuration and operational state compliance
- The `template` module to generate reports about configuration and operational state information
- Templates and `command` or `config` modules to generate host, device, or platform commands or configuration
- The current platform `facts` modules to supplement native facts information

By parsing semi-structured text into Ansible native data structures, you can take full advantage of Ansible's network modules and plugins.

When not to parse the text

You should not parse semi-structured text when:

- The device, host, or platform has a RESTAPI and returns JSON.
- Existing Ansible facts modules already return the desired data.
- Ansible network resource modules exist for configuration management of the device and resource.

Parsing the CLI

The `cli_parse` module includes the following cli_parsing plugins:

`native`
: The native parsing engine built into Ansible and requires no addition python libraries

`xml`
: Convert XML to an Ansible native data structure

`textfsm`
: A python module which implements a template based state machine for parsing semi-formatted text

`ntc_templates`
: Predefined `textfsm` templates packages supporting a variety of platforms and commands

`ttp`
: A library for semi-structured text parsing using templates, with added capabilities to simplify the process

`pyats`
: Uses the parsers included with the Cisco Test Automation & Validation Solution

`jc`
: A python module that converts the output of dozens of popular Linux/UNIX/macOS/Windows commands and file types to python dictionaries or lists of dictionaries. Note: this filter plugin can be found in the `community.general` collection.

`json`
: Converts JSON output at the CLI to an Ansible native data structure

Although Ansible contains a number of plugins that can convert XML to Ansible native data structures, the `cli_parse` module runs the command on devices that return XML and returns the converted data in a single task.

Because `cli_parse` uses a plugin based architecture, it can use additional parsing engines from any Ansible collection.

Note: The `ansible.netcommon.native` and `ansible.utils.json` parsing engines are fully supported with a Red Hat Ansible Automation Platform subscription. Red Hat Ansible Automation Platform subscription support is limited to the use of the `ntc_templates`, pyATS, `textfsm`, `xmltodict`, public APIs as documented.

Parsing with the native parsing engine

The native parsing engine is included with the `cli_parse` module. It uses data captured using regular expressions to populate the parsed data structure. The native parsing engine requires a YAML template file to parse the command output.

Networking example

This example uses the output of a network device command and applies a native template to produce an output in Ansible structured data format.

The `show interface` command output from the network device looks as follows:

```
Ethernet1/1 is up
admin state is up, Dedicated Interface
  Hardware: 100/1000/10000 Ethernet, address: 5254.005a.f8bd (bia 5254.005a.f8bd)
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, medium is broadcast
  Port mode is access
  full-duplex, auto-speed
  Beacon is turned off
  Auto-Negotiation is turned on  FEC mode is Auto
  Input flow-control is off, output flow-control is off
  Auto-mdix is turned off
  Switchport monitor is off
  EtherType is 0x8100
  EEE (efficient-ethernet) : n/a
  Last link flapped 4week(s) 6day(s)
  Last clearing of "show interface" counters never
<...>
```

Create the native template to match this output and store it as `templates/nxos_show_interface.yaml`:

```yaml
---
- example: Ethernet1/1 is up
  getval: '(?P<name>\S+) is (?P<oper_state>\S+)'
  result:
    "{{ name }}":
      name: "{{ name }}"
      state:
        operating: "{{ oper_state }}"
  shared: true

- example: admin state is up, Dedicated Interface
  getval: 'admin state is (?P<admin_state>\S+),'
  result:
    "{{ name }}":
      name: "{{ name }}"
      state:
        admin: "{{ admin_state }}"

- example: "  Hardware: Ethernet, address: 5254.005a.f8b5 (bia 5254.005a.f8b5)"
```

(continues on next page)

(continued from previous page)

```
  getval: '\s+Hardware: (?P<hardware>.*), address: (?P<mac>\S+)'
  result:
    "{{ name }}":
      hardware: "{{ hardware }}"
      mac_address: "{{ mac }}"
```

This native parser template is structured as a list of parsers, each containing the following key-value pairs:

- `example` - An example line of the text line to be parsed
- `getval` - A regular expression using named capture groups to store the extracted data
- `result` - A data tree, populated as a template, from the parsed data
- `shared` - (optional) The shared key makes the parsed values available to the rest of the parser entries until matched again.

The following example task uses `cli_parse` with the native parser and the example template above to parse the `show interface` command from a Cisco NXOS device:

```
- name: "Run command and parse with native"
  ansible.utils.cli_parse:
    command: show interface
    parser:
      name: ansible.netcommon.native
    set_fact: interfaces
```

Taking a deeper dive into this task:

- The `command` option provides the command you want to run on the device or host. Alternately, you can provide text from a previous command with the `text` option instead.
- The `parser` option provides information specific to the parser engine.
- The `name` suboption provides the fully qualified collection name (FQCN) of the parsing engine (`ansible.netcommon.native`).
- The `cli_parse` module, by default, looks for the template in the templates directory as `{{ short_os }}_{{ command }}.yaml`.
  - The `short_os` in the template filename is derived from either the host `ansible_network_os` or `ansible_distribution`.
  - Spaces in the network or host command are replace with `_` in the `command` portion of the template filename. In this example, the `show interfaces` network CLI command becomes `show_interfaces` in the filename.

Note: `ansible.netcommon.native` parsing engine is fully supported with a Red Hat Ansible Automation Platform subscription.

Lastly in this task, the `set_fact` option sets the following `interfaces` fact for the device based on the now-structured data returned from `cli_parse`:

```
Ethernet1/1:
    hardware: 100/1000/10000 Ethernet
    mac_address: 5254.005a.f8bd
    name: Ethernet1/1
```

(continues on next page)

(continued from previous page)

```
    state:
    admin: up
    operating: up
Ethernet1/10:
    hardware: 100/1000/10000 Ethernet
    mac_address: 5254.005a.f8c6
<...>
```

Linux example

You can also use the native parser to run commands and parse output from Linux hosts.

The output of a sample Linux command (`ip addr show`) looks as follows:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen␣
↪1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN␣
↪group default qlen 1000
    link/ether x2:6a:64:9d:84:19 brd ff:ff:ff:ff:ff:ff
3: wlp2s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether x6:c2:44:f7:41:e0 brd ff:ff:ff:ff:ff:ff permaddr d8:f2:ca:99:5c:82
```

Create the native template to match this output and store it as `templates/fedora_ip_addr_show.yaml`:

```
---
- example: '1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group␣
↪default qlen 1000'
  getval: |
    (?x)                                                # free-spacing
    \d+:\s                                              # the interface index
    (?P<name>\S+):\s                                    # the name
    <(?P<properties>\S+)>                               # the properties
    \smtu\s(?P<mtu>\d+)                                 # the mtu
    .*                                                  # gunk
    state\s(?P<state>\S+)                               # the state of the interface
  result:
    "{{ name }}":
        name: "{{ name }}"
        loopback: "{{ 'LOOPBACK' in stats.split(',') }}"
        up: "{{ 'UP' in properties.split(',')  }}"
        carrier: "{{ not 'NO-CARRIER' in properties.split(',') }}"
        broadcast: "{{ 'BROADCAST' in properties.split(',') }}"
        multicast: "{{ 'MULTICAST' in properties.split(',') }}"
        state: "{{ state|lower() }}"
        mtu: "{{ mtu }}"
  shared: True
```

(continues on next page)

(continued from previous page)

```
- example: 'inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0'
  getval: |
   (?x)                                                  # free-spacing
   \s+inet\s(?P<inet>([0-9]{1,3}\.){3}[0-9]{1,3})       # the ip address
   /(?P<bits>\d{1,2})                                    # the mask bits
  result:
    "{{ name }}":
        ip_address: "{{ inet }}"
        mask_bits: "{{ bits }}"
```

Note: The shared key in the parser template allows the interface name to be used in subsequent parser entries. The use of examples and free-spacing mode with the regular expressions makes the template easier to read.

The following example task uses cli_parse with the native parser and the example template above to parse the Linux output:

```
- name: Run command and parse
  ansible.utils.cli_parse:
    command: ip addr show
    parser:
      name: ansible.netcommon.native
    set_fact: interfaces
```

This task assumes you previously gathered facts to determine the ansible_distribution needed to locate the template. Alternately, you could provide the path in the parser/template_path option.

Lastly in this task, the set_fact option sets the following interfaces fact for the host, based on the now-structured data returned from cli_parse:

```
lo:
  broadcast: false
  carrier: true
  ip_address: 127.0.0.1
  mask_bits: 8
  mtu: 65536
  multicast: false
  name: lo
  state: unknown
  up: true
enp64s0u1:
  broadcast: true
  carrier: true
  ip_address: 192.168.86.83
  mask_bits: 24
  mtu: 1500
  multicast: true
  name: enp64s0u1
  state: up
  up: true
<...>
```

Parsing JSON

Although Ansible will natively convert serialized JSON to Ansible native data when recognized, you can also use the `cli_parse` module for this conversion.

Example task:

```yaml
- name: "Run command and parse as json"
  ansible.utils.cli_parse:
    command: show interface | json
    parser:
      name: ansible.utils.json
    register: interfaces
```

Taking a deeper dive into this task:

- The `show interface | json` command is issued on the device.
- The output is set as the `interfaces` fact for the device.
- JSON support is provided primarily for playbook consistency.

Note: The use of `ansible.netcommon.json` is fully supported with a Red Hat Ansible Automation Platform subscription

Parsing with ntc_templates

The `ntc_templates` python library includes pre-defined `textfsm` templates for parsing a variety of network device commands output.

Example task:

```yaml
- name: "Run command and parse with ntc_templates"
  ansible.utils.cli_parse:
    command: show interface
    parser:
      name: ansible.netcommon.ntc_templates
    set_fact: interfaces
```

Taking a deeper dive into this task:

- The `ansible_network_os` of the device is converted to the ntc_template format `cisco_nxos`. Alternately, you can provide the `os` with the `parser/os` option instead.
- The `cisco_nxos_show_interface.textfsm` template, included with the `ntc_templates` package, parses the output.
- See the ntc_templates README for additional information about the `ntc_templates` python library.

Note: Red Hat Ansible Automation Platform subscription support is limited to the use of the `ntc_templates` public APIs as documented.

This task and and the predefined template sets the following fact as the `interfaces` fact for the host:

```
interfaces:
- address: 5254.005a.f8b5
  admin_state: up
  bandwidth: 1000000 Kbit
  bia: 5254.005a.f8b5
  delay: 10 usec
  description: ''
  duplex: full-duplex
  encapsulation: ARPA
  hardware_type: Ethernet
  input_errors: ''
  input_packets: ''
  interface: mgmt0
  ip_address: 192.168.101.14/24
  last_link_flapped: ''
  link_status: up
  mode: ''
  mtu: '1500'
  output_errors: ''
  output_packets: ''
  speed: 1000 Mb/s
- address: 5254.005a.f8bd
  admin_state: up
  bandwidth: 1000000 Kbit
  bia: 5254.005a.f8bd
  delay: 10 usec
```

Parsing with pyATS

pyATS is part of the Cisco Test Automation & Validation Solution. It includes many predefined parsers for a number of network platforms and commands. You can use the predefined parsers that are part of the pyATS package with the cli_parse module.

Example task:

```
- name: "Run command and parse with pyats"
  ansible.utils.cli_parse:
    command: show interface
    parser:
      name: ansible.netcommon.pyats
    set_fact: interfaces
```

Taking a deeper dive into this task:

- The cli_parse modules converts the ansible_network_os automatically (in this example, ansible_network_os set to cisco.nxos.nxos, converts to nxos for pyATS. Alternately, you can set the OS with the parser/os option instead.
- Using a combination of the command and OS, the pyATS selects the following parser: https://pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/parsers/show%2520interface.
- The cli_parse module sets cisco.ios.ios to iosxe for pyATS. You can override this with the parser/os option.

- `cli_parse` only uses the predefined parsers in pyATS. See the pyATS documentation and the full list of pyATS included parsers.

Note: Red Hat Ansible Automation Platform subscription support is limited to the use of the pyATS public APIs as documented.

This task sets the following fact as the `interfaces` fact for the host:

```
mgmt0:
  admin_state: up
  auto_mdix: 'off'
  auto_negotiate: true
  bandwidth: 1000000
  counters:
    in_broadcast_pkts: 3
    in_multicast_pkts: 1652395
    in_octets: 556155103
    in_pkts: 2236713
    in_unicast_pkts: 584259
    rate:
      in_rate: 320
      in_rate_pkts: 0
      load_interval: 1
      out_rate: 48
      out_rate_pkts: 0
    rx: true
    tx: true
  delay: 10
  duplex_mode: full
  enabled: true
  encapsulations:
    encapsulation: arpa
  ethertype: '0x0000'
  ipv4:
    192.168.101.14/24:
      ip: 192.168.101.14
      prefix_length: '24'
  link_state: up
  <...>
```

Parsing with textfsm

`textfsm` is a Python module which implements a template-based state machine for parsing semi-formatted text.

The following sample `textfsm` template is stored as `templates/nxos_show_interface.textfsm`

```
Value Required INTERFACE (\S+)
Value LINK_STATUS (.+?)
Value ADMIN_STATE (.+?)
Value HARDWARE_TYPE (.\*)
Value ADDRESS ([a-zA-Z0-9]+.[a-zA-Z0-9]+.[a-zA-Z0-9]+)
Value BIA ([a-zA-Z0-9]+.[a-zA-Z0-9]+.[a-zA-Z0-9]+)
```

(continues on next page)

(continued from previous page)

```
Value DESCRIPTION (.\*)
Value IP_ADDRESS (\d+\.\d+\.\d+\.\d+\/\d+)
Value MTU (\d+)
Value MODE (\S+)
Value DUPLEX (.+duplex?)
Value SPEED (.+?)
Value INPUT_PACKETS (\d+)
Value OUTPUT_PACKETS (\d+)
Value INPUT_ERRORS (\d+)
Value OUTPUT_ERRORS (\d+)
Value BANDWIDTH (\d+\s+\w+)
Value DELAY (\d+\s+\w+)
Value ENCAPSULATION (\w+)
Value LAST_LINK_FLAPPED (.+?)

Start
  ^\S+\s+is.+ -> Continue.Record
  ^${INTERFACE}\s+is\s+${LINK_STATUS},\sline\sprotocol\sis\s${ADMIN_STATE}$$
  ^${INTERFACE}\s+is\s+${LINK_STATUS}$$
  ^admin\s+state\s+is\s+${ADMIN_STATE},
  ^\s+Hardware(:|\s+is)\s+${HARDWARE_TYPE},\s+address(:|\s+is)\s+${ADDRESS}(.*bia\s+$
↪{BIA})*
  ^\s+Description:\s+${DESCRIPTION}
  ^\s+Internet\s+Address\s+is\s+${IP_ADDRESS}
  ^\s+Port\s+mode\s+is\s+${MODE}
  ^\s+${DUPLEX}, ${SPEED}(,|$$)
  ^\s+MTU\s+${MTU}.\*BW\s+${BANDWIDTH}.\*DLY\s+${DELAY}
  ^\s+Encapsulation\s+${ENCAPSULATION}
  ^\s+${INPUT_PACKETS}\s+input\s+packets\s+\d+\s+bytes\s\*$$
  ^\s+${INPUT_ERRORS}\s+input\s+error\s+\d+\s+short\s+frame\s+\d+\s+overrun\s+\d+\
↪s+underrun\s+\d+\s+ignored\s\*$$
  ^\s+${OUTPUT_PACKETS}\s+output\s+packets\s+\d+\s+bytes\s\*$$
  ^\s+${OUTPUT_ERRORS}\s+output\s+error\s+\d+\s+collision\s+\d+\s+deferred\s+\d+\s+late\
↪s+collision\s\*$$
  ^\s+Last\s+link\s+flapped\s+${LAST_LINK_FLAPPED}\s\*$$
```

The following task uses the example template for `textfsm` with the `cli_parse` module.

```yaml
- name: "Run command and parse with textfsm"
  ansible.utils.cli_parse:
    command: show interface
    parser:
      name: ansible.utils.textfsm
    set_fact: interfaces
```

Taking a deeper dive into this task:

- The `ansible_network_os` for the device (`cisco.nxos.nxos`) is converted to `nxos`. Alternately you can provide the OS in the `parser/os` option instead.
- The textfsm template name defaulted to `templates/nxos_show_interface.textfsm` using a combination of the OS and command run. Alternately you can override the generated template path with the `parser/template_path` option.

- See the textfsm README for details.
- textfsm was previously made available as a filter plugin. Ansible users should transition to the cli_parse module.

Note: Red Hat Ansible Automation Platform subscription support is limited to the use of the textfsm public APIs as documented.

This task sets the following fact as the interfaces fact for the host:

```
- ADDRESS: X254.005a.f8b5
  ADMIN_STATE: up
  BANDWIDTH: 1000000 Kbit
  BIA: X254.005a.f8b5
  DELAY: 10 usec
  DESCRIPTION: ''
  DUPLEX: full-duplex
  ENCAPSULATION: ARPA
  HARDWARE_TYPE: Ethernet
  INPUT_ERRORS: ''
  INPUT_PACKETS: ''
  INTERFACE: mgmt0
  IP_ADDRESS: 192.168.101.14/24
  LAST_LINK_FLAPPED: ''
  LINK_STATUS: up
  MODE: ''
  MTU: '1500'
  OUTPUT_ERRORS: ''
  OUTPUT_PACKETS: ''
  SPEED: 1000 Mb/s
- ADDRESS: X254.005a.f8bd
  ADMIN_STATE: up
  BANDWIDTH: 1000000 Kbit
  BIA: X254.005a.f8bd
```

Parsing with TTP

TTP is a Python library for semi-structured text parsing using templates. TTP uses a jinja-like syntax to limit the need for regular expressions. Users familiar with jinja templating may find the TTP template syntax familiar.

The following is an example TTP template stored as templates/nxos_show_interface.ttp:

```
{{ interface }} is {{ state }}
admin state is {{ admin_state }}{{ ignore(".\*") }}
```

The following task uses this template to parse the show interface command output:

```
- name: "Run command and parse with ttp"
  ansible.utils.cli_parse:
    command: show interface
    parser:
      name: ansible.utils.ttp
    set_fact: interfaces
```

Taking a deeper dive in this task:

- The default template path `templates/nxos_show_interface.ttp` was generated using the `ansible_network_os` for the host and `command` provided.
- TTP supports several additional variables that will be passed to the parser. These include:
  - `parser/vars/ttp_init` - Additional parameter passed when the parser is initialized.
  - `parser/vars/ttp_results` - Additional parameters used to influence the parser output.
  - `parser/vars/ttp_vars` - Additional variables made available in the template.
- See the TTP documentation for details.

The task sets the follow fact as the `interfaces` fact for the host:

```
- admin_state: up,
  interface: mgmt0
  state: up
- admin_state: up,
  interface: Ethernet1/1
  state: up
- admin_state: up,
  interface: Ethernet1/2
  state: up
```

Parsing with JC

JC is a python library that converts the output of dozens of common Linux/UNIX/macOS/Windows command-line tools and file types to python dictionaries or lists of dictionaries for easier parsing. JC is available as a filter plugin in the `community.general` collection.

The following is an example using JC to parse the output of the `dig` command:

```
- name: "Run dig command and parse with jc"
  hosts: ubuntu
  tasks:
  - shell: dig example.com
    register: result
  - set_fact:
      myvar: "{{ result.stdout | community.general.jc('dig') }}"
  - debug:
      msg: "The IP is: {{ myvar[0].answer[0].data }}"
```

- The JC project and documentation can be found here.
- See this blog entry for more information.

Converting XML

Although Ansible contains a number of plugins that can convert XML to Ansible native data structures, the `cli_parse` module runs the command on devices that return XML and returns the converted data in a single task.

This example task runs the `show interface` command and parses the output as XML:

```
- name: "Run command and parse as xml"
    ansible.utils.cli_parse:
      command: show interface | xml
      parser:
        name: ansible.utils.xml
  set_fact: interfaces
```

Note: Red Hat Ansible Automation Platform subscription support is limited to the use of the `xmltodict` public APIs as documented.

This task sets the `interfaces` fact for the host based on this returned output:

```
nf:rpc-reply:
  '@xmlns': http://www.cisco.com/nxos:1.0:if_manager
  '@xmlns:nf': urn:ietf:params:xml:ns:netconf:base:1.0
  nf:data:
    show:
      interface:
        __XML__OPT_Cmd_show_interface_quick:
          __XML__OPT_Cmd_show_interface___readonly__:
            __readonly__:
              TABLE_interface:
                ROW_interface:
                - admin_state: up
                  encapsulation: ARPA
                  eth_autoneg: 'on'
                  eth_bia_addr: x254.005a.f8b5
                  eth_bw: '1000000'
```

Advanced use cases

The `cli_parse` module supports several features to support more complex uses cases.

Provide a full template path

Use the `template_path` option to override the default template path in the task:

```
- name: "Run command and parse with native"
  ansible.utils.cli_parse:
    command: show interface
    parser:
      name: ansible.netcommon.native
      template_path: /home/user/templates/filename.yaml
```

Provide command to parser different than the command run

Use the command suboption for the parser to configure the command the parser expects if it is different from the command cli_parse runs:

```
- name: "Run command and parse with native"
  ansible.utils.cli_parse:
    command: sho int
    parser:
      name: ansible.netcommon.native
      command: show interface
```

Provide a custom OS value

Use the os suboption to the parser to directly set the OS instead of using ansible_network_os or ansible_distribution to generate the template path or with the specified parser engine:

```
- name: Use ios instead of iosxe for pyats
  ansible.utils.cli_parse:
    command: show something
    parser:
      name: ansible.netcommon.pyats
      os: ios

- name: Use linux instead of fedora from ansible_distribution
  ansible.utils.cli_parse:
    command: ps -ef
    parser:
      name: ansible.netcommon.native
      os: linux
```

Parse existing text

Use the text option instead of command to parse text collected earlier in the playbook.

```
# using /home/user/templates/filename.yaml
- name: "Parse text from previous task"
  ansible.utils.cli_parse:
    text: "{{ output['stdout'] }}"
    parser:
      name: ansible.netcommon.native
      template_path: /home/user/templates/filename.yaml

 # using /home/user/templates/filename.yaml
- name: "Parse text from file"
  ansible.utils.cli_parse:
    text: "{{ lookup('file', 'path/to/file.txt') }}"
    parser:
      name: ansible.netcommon.native
      template_path: /home/user/templates/filename.yaml
```

(continues on next page)

(continued from previous page)

```
# using templates/nxos_show_version.yaml
- name: "Parse text from previous task"
  ansible.utils.cli_parse:
    text: "{{ sho_version['stdout'] }}"
    parser:
      name: ansible.netcommon.native
      os: nxos
      command: show version
```

See also:

- *Developing cli_parser plugins in a collection*

1.22.4 Validate data against set criteria with Ansible

The validate module validates data against your predefined criteria using a validation engine. You can pull this data from a device or file, validate it against your defined criteria, and use the results to identify configuration or operational state drift and optionally take remedial action.

Understanding the validate plugin

The ansible.utils collection includes the validate module.

To validate data:

1. Pull in structured data or convert your data to structured format with the cli_parse module.
2. Define the criteria to test that data against.
3. Select a validation engine and test the data to see if it is valid based on the selected criteria and validation engine.

The structure of the data and the criteria depends on the validation engine you select. The examples here use the `jsonschema` validation engine provided in the ansible.utils collection.Red Hat Ansible Automation Platform subscription supports limited use if jsonschema public APIs as documented.

Structuring the data

You can pull previously structured data from a file, or use the cli_parse module to structure your data.

The following example fetches the operational state of some network (Cisco NXOS) interfaces and translates that state to structured data using the `ansible.netcommon.pyats` parser.

```
---
- hosts: nxos
  connection: ansible.netcommon.network_cli
  gather_facts: false
  vars:
    ansible_network_os: cisco.nxos.nxos
    ansible_user: "changeme"
    ansible_password: "changeme"

  tasks:
  - name: "Fetch interface state and parse with pyats"
    ansible.utils.cli_parse:
      command: show interface
      parser:
        name: ansible.netcommon.pyats
    register: nxos_pyats_show_interface

  - name: print structured interface state data
    ansible.builtin.debug:
      msg: "{{ nxos_pyats_show_interface['parsed'] }}"
----
```

This results in the following structured data.

```
ok: [nxos] => {
"changed": false,
"parsed": {
    "Ethernet2/1": {
        "admin_state": "down",
        "auto_mdix": "off",
        "auto_negotiate": false,
        "bandwidth": 1000000,
        "beacon": "off"
        <--output omitted-->
    },
    "Ethernet2/10": {
        "admin_state": "down",
        "auto_mdix": "off",
        "auto_negotiate": false,
        "bandwidth": 1000000,
        "beacon": "off",
        <--output omitted-->
    }
  }
}
```

See *Parsing semi-structured text with Ansible* for details on how to parse semi-structured data into structured data.

Defining the criteria to validate against

This example uses the jsonschema validation engine to parse the JSON structured data we created in the prior section. the criteria defines the state we want the data to conform to. In this instance, we can validate against a desired admin state of up for all the interfaces.

The criteria for `jsonschema` in this example is as follows:

```
$cat criteria/nxos_show_interface_admin_criteria.json
{
      "type" : "object",
      "patternProperties": {
              "^.*": {
                      "type": "object",
                      "properties": {
                              "admin_state": {
                                      "type": "string",
                                      "pattern": "up"
                              }
                      }
              }
      }
 }
```

Validating the data

Now that we have the structured data and the criteria, we can validate this data with the validate module.

The following tasks check if the current state of the interfaces match the desired state defined in the criteria file.

```
- name: Validate interface admin state
  ansible.utils.validate:
    data: "{{ nxos_pyats_show_interface['parsed'] }}"
    criteria:
      - "{{ lookup('file',  './criteria/nxos_show_interface_admin_criteria.json') | from_
↪json }}"
    engine: ansible.utils.jsonschema
  ignore_errors: true
  register: result

- name: Print the interface names that do not satisfy the desired state
  ansible.builtin.debug:
    msg: "{{ item['data_path'].split('.')[0] }}"
  loop: "{{ result['errors'] }}"
  when: "'errors' in result"
```

In these tasks, we have:

1. Set `data` to the structured JSON data from the cli_parse module.
2. Set `criteria` to the JSON criteria file we defined.
3. Set the validate engine to `jsonschema`.

Note: The value of the criteria option can be a list and should be in a format that is defined by the validation engine used. You need to install the jsonschema on the control node for this example.

The tasks output a list of errors indicating interfaces that do not have admin value in up state.

```
TASK [Validate interface for admin state]␣
↪*****************************************************************************************************
fatal: [nxos02]: FAILED! => {"changed": false, "errors": [{"data_path": "Ethernet2/1.
↪admin_state", "expected": "up", "found": "down", "json_path": "$.Ethernet2/1.admin_
↪state", "message": "'down' does not match 'up'", "relative_schema": {"pattern": "up",
↪"type": "string"}, "schema_path": "patternProperties.^.*.properties.admin_state.pattern
↪", "validator": "pattern"}, {"data_path": "Ethernet2/10.admin_state", "expected": "up",
↪ "found": "down", "json_path": "$.Ethernet2/10.admin_state", "message": "'down' does␣
↪not match 'up'", "relative_schema": {"pattern": "up", "type": "string"}, "schema_path
↪": "patternProperties.^.*.properties.admin_state.pattern", "validator": "pattern"}],
↪"msg": "Validation errors were found.\nAt 'patternProperties.^.*.properties.admin_
↪state.pattern' 'down' does not match 'up'. \nAt 'patternProperties.^.*.properties.
↪admin_state.pattern' 'down' does not match 'up'. \nAt 'patternProperties.^.*.
↪properties.admin_state.pattern' 'down' does not match 'up'. "}
...ignoring


TASK [Print the interface names that do not satisfy the desired state]␣
↪****************************************************************************
Monday 14 December 2020  11:05:38 +0530 (0:00:01.661)       0:00:28.676 *******
ok: [nxos] => {
   "msg": "Ethernet2/1"
}
ok: [nxos] => {
   "msg": "Ethernet2/10"
}
```

This shows Ethernet2/1 and Ethernet2/10 are not in the desired state based on the defined criteria. You can create a report or take further action to remediate this to bring the interfaces to the desired state based on the defined criteria.

1.22.5 Network Debug and Troubleshooting Guide

This section discusses how to debug and troubleshoot network modules in Ansible.

How to troubleshoot

Ansible network automation errors generally fall into one of the following categories:

Authentication issues

- Not correctly specifying credentials
- Remote device (network switch/router) not falling back to other other authentication methods
- SSH key issues

Timeout issues

- Can occur when trying to pull a large amount of data
- May actually be masking a authentication issue

Playbook issues

- Use of `delegate_to`, instead of `ProxyCommand`. See *network proxy guide* for more information.

Warning: `unable to open shell`

The `unable to open shell` message means that the `ansible-connection` daemon has not been able to successfully talk to the remote network device. This generally means that there is an authentication issue. See the "Authentication and connection issues" section in this document for more information.

Enabling Networking logging and how to read the logfile

Platforms: Any

Ansible includes logging to help diagnose and troubleshoot issues regarding Ansible Networking modules.

Because logging is very verbose, it is disabled by default. It can be enabled with the *ANSIBLE_LOG_PATH* and *ANSIBLE_DEBUG* options on the ansible-controller, that is the machine running `ansible-playbook`.

Before running `ansible-playbook`, run the following commands to enable logging:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/ansible.log
# Enable Debug
export ANSIBLE_DEBUG=True

# Run with 4*v for connection level verbosity
ansible-playbook -vvvv ...
```

After Ansible has finished running you can inspect the log file which has been created on the ansible-controller:

```
less $ANSIBLE_LOG_PATH

2017-03-30 13:19:52,740 p=28990 u=fred |  creating new control socket for host veos01:22␣
↪as user admin
2017-03-30 13:19:52,741 p=28990 u=fred |  control socket path is /home/fred/.ansible/pc/
↪ca5960d27a
2017-03-30 13:19:52,741 p=28990 u=fred |  current working directory is /home/fred/
↪ansible/test/integration
2017-03-30 13:19:52,741 p=28990 u=fred |  using connection plugin network_cli
...
2017-03-30 13:20:14,771 paramiko.transport userauth is OK
2017-03-30 13:20:15,283 paramiko.transport Authentication (keyboard-interactive)␣
↪successful!
2017-03-30 13:20:15,302 p=28990 u=fred |  ssh connection done, setting terminal
2017-03-30 13:20:15,321 p=28990 u=fred |  ssh connection has completed successfully
2017-03-30 13:20:15,322 p=28990 u=fred |  connection established to veos01 in 0:00:22.
↪580626
```

From the log notice:

- `p=28990` Is the PID (Process ID) of the `ansible-connection` process
- `u=fred` Is the user *running* ansible, not the remote-user you are attempting to connect as
- `creating new control socket for host veos01:22 as user admin` host:port as user
- `control socket path is` location on disk where the persistent connection socket is created
- `using connection plugin network_cli` Informs you that persistent connection is being used

- `connection established to veos01 in 0:00:22.580626` Time taken to obtain a shell on the remote device

Note: Port None `creating new control socket for host veos01:None`

If the log reports the port as `None` this means that the default port is being used. A future Ansible release will improve this message so that the port is always logged.

Because the log files are verbose, you can use grep to look for specific information. For example, once you have identified the `pid` from the `creating new control socket for host` line you can search for other connection log entries:

```
grep "p=28990" $ANSIBLE_LOG_PATH
```

Enabling Networking device interaction logging

Platforms: Any

Ansible includes logging of device interaction in the log file to help diagnose and troubleshoot issues regarding Ansible Networking modules. The messages are logged in the file pointed to by the `log_path` configuration option in the Ansible configuration file or by setting the *ANSIBLE_LOG_PATH*.

Warning: The device interaction messages consist of command executed on the target device and the returned response. Since this log data can contain sensitive information including passwords in plain text it is disabled by default. Additionally, in order to prevent accidental leakage of data, a warning will be shown on every task with this setting enabled, specifying which host has it enabled and where the data is being logged.

Be sure to fully understand the security implications of enabling this option. The device interaction logging can be enabled either globally by setting in configuration file or by setting environment or enabled on per task basis by passing a special variable to the task.

Before running `ansible-playbook` run the following commands to enable logging:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/ansible.log
```

Enable device interaction logging for a given task

```
- name: get version information
  cisco.ios.ios_command:
    commands:
      - show version
  vars:
    ansible_persistent_log_messages: True
```

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
log_messages = True
```

or enable the environment variable *ANSIBLE_PERSISTENT_LOG_MESSAGES*:

```
# Enable device interaction logging
export ANSIBLE_PERSISTENT_LOG_MESSAGES=True
```

If the task is failing on connection initialization itself, you should enable this option globally. If an individual task is failing intermittently this option can be enabled for that task itself to find the root cause.

After Ansible has finished running you can inspect the log file which has been created on the ansible-controller

Note: Be sure to fully understand the security implications of enabling this option as it can log sensitive information in log file thus creating security vulnerability.

Isolating an error

Platforms: Any

As with any effort to troubleshoot it's important to simplify the test case as much as possible.

For Ansible this can be done by ensuring you are only running against one remote device:

- Using `ansible-playbook --limit switch1.example.net...`
- Using an ad hoc `ansible` command

ad hoc refers to running Ansible to perform some quick command using `/usr/bin/ansible`, rather than the orchestration language, which is `/usr/bin/ansible-playbook`. In this case we can ensure connectivity by attempting to execute a single command on the remote device:

```
ansible -m arista.eos.eos_command -a 'commands=?' -i inventory switch1.example.net -e
↪'ansible_connection=ansible.netcommon.network_cli' -u admin -k
```

In the above example, we:

- connect to `switch1.example.net` specified in the inventory file `inventory`
- use the module `arista.eos.eos_command`
- run the command `?`
- connect using the username `admin`
- inform the `ansible` command to prompt for the SSH password by specifying `-k`

If you have SSH keys configured correctly, you don't need to specify the `-k` parameter.

If the connection still fails you can combine it with the enable_network_logging parameter. For example:

```
# Specify the location for the log file
export ANSIBLE_LOG_PATH=~/ansible.log
# Enable Debug
export ANSIBLE_DEBUG=True
# Run with ``-vvvv`` for connection level verbosity
ansible -m arista.eos.eos_command -a 'commands=?' -i inventory switch1.example.net -e
↪'ansible_connection=ansible.netcommon.network_cli' -u admin -k
```

Then review the log file and find the relevant error message in the rest of this document.

Troubleshooting socket path issues

Platforms: Any

The `Socket path does not exist or cannot be found` and `Unable to connect to socket` messages indicate that the socket used to communicate with the remote network device is unavailable or does not exist.

For example:

```
fatal: [spine02]: FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_TSqk5J/
↪ansible_modlib.zip/ansible/module_utils/connection.py\", line 115, in _exec_jsonrpc\
↪nansible.module_utils.connection.ConnectionError: Socket path XX does not exist or␣
↪cannot be found. See Troubleshooting socket path issues in the Network Debug and␣
↪Troubleshooting Guide\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 1
}
```

or

```
fatal: [spine02]: FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_TSqk5J/
↪ansible_modlib.zip/ansible/module_utils/connection.py\", line 123, in _exec_jsonrpc\
↪nansible.module_utils.connection.ConnectionError: Unable to connect to socket XX. See␣
↪Troubleshooting socket path issues in Network Debug and Troubleshooting Guide\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE",
    "rc": 1
}
```

Suggestions to resolve:

1. Verify that you have write access to the socket path described in the error message.
2. Follow the steps detailed in *enable network logging*.

If the identified error message from the log file is:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  command timeout triggered, timeout value is 30␣
↪secs
```

or

```
2017-04-04 12:19:05,670 p=18591 u=fred |  persistent connection idle timeout triggered,␣
↪timeout value is 30 secs
```

Follow the steps detailed in *timeout issues*

Category "Unable to open shell"

Platforms: Any

The `unable to open shell` message means that the `ansible-connection` daemon has not been able to successfully talk to the remote network device. This generally means that there is an authentication issue. It is a "catch all" message, meaning you need to enable logging to find the underlying issues.

For example:

```
TASK [prepare_eos_tests : enable cli on remote device]␣
↪*****************************************************
fatal: [veos01]: FAILED! => {"changed": false, "failed": true, "msg": "unable to open␣
↪shell"}
```

or:

```
TASK [ios_system : configure name_servers]␣
↪****************************************************************
task path:
fatal: [ios-csr1000v]: FAILED! => {
    "changed": false,
    "failed": true,
    "msg": "unable to open shell",
}
```

Suggestions to resolve:

Follow the steps detailed in *enable_network_logging*.

Once you've identified the error message from the log file, the specific solution can be found in the rest of this document.

Error: "[Errno -2] Name or service not known"

Platforms: Any

Indicates that the remote host you are trying to connect to can not be reached

For example:

```
2017-04-04 11:39:48,147 p=15299 u=fred |  control socket path is /home/fred/.ansible/pc/
↪ca5960d27a
2017-04-04 11:39:48,147 p=15299 u=fred |  current working directory is /home/fred/git/
↪ansible-inc/stable-2.3/test/integration
2017-04-04 11:39:48,147 p=15299 u=fred |  using connection plugin network_cli
2017-04-04 11:39:48,340 p=15299 u=fred |  connecting to host veos01 returned an error
2017-04-04 11:39:48,340 p=15299 u=fred |  [Errno -2] Name or service not known
```

Suggestions to resolve:

- If you are using the `provider:` options ensure that its suboption `host:` is set correctly.
- If you are not using `provider:` nor top-level arguments ensure your inventory file is correct.

Error: "Authentication failed"

Platforms: Any

Occurs if the credentials (username, passwords, or ssh keys) passed to `ansible-connection` (through `ansible` or `ansible-playbook`) can not be used to connect to the remote device.

For example:

```
<ios01> ESTABLISH CONNECTION FOR USER: cisco on PORT 22 TO ios01
<ios01> Authentication failed.
```

Suggestions to resolve:

If you are specifying credentials through `password:` (either directly or through `provider:`) or the environment variable *ANSIBLE_NET_PASSWORD* it is possible that `paramiko` (the Python SSH library that Ansible uses) is using ssh keys, and therefore the credentials you are specifying are being ignored. To find out if this is the case, disable "look for keys". This can be done like this:

```
export ANSIBLE_PARAMIKO_LOOK_FOR_KEYS=False
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[paramiko_connection]
look_for_keys = False
```

Error: "connecting to host <hostname> returned an error" or "Bad address"

This may occur if the SSH fingerprint hasn't been added to Paramiko's (the Python SSH library) know hosts file.

When using persistent connections with Paramiko, the connection runs in a background process. If the host doesn't already have a valid SSH key, by default Ansible will prompt to add the host key. This will cause connections running in background processes to fail.

For example:

```
2017-04-04 12:06:03,486 p=17981 u=fred |  using connection plugin network_cli
2017-04-04 12:06:04,680 p=17981 u=fred |  connecting to host veos01 returned an error
2017-04-04 12:06:04,682 p=17981 u=fred |  (14, 'Bad address')
2017-04-04 12:06:33,519 p=17981 u=fred |  number of connection attempts exceeded, unable␣
↪to connect to control socket
2017-04-04 12:06:33,520 p=17981 u=fred |  persistent_connect_interval=1, persistent_
↪connect_retries=30
```

Suggestions to resolve:

Use `ssh-keyscan` to pre-populate the known_hosts. You need to ensure the keys are correct.

```
ssh-keyscan veos01
```

or

You can tell Ansible to automatically accept the keys

Environment variable method:

```
export ANSIBLE_PARAMIKO_HOST_KEY_AUTO_ADD=True
ansible-playbook ...
```

`ansible.cfg` method:

ansible.cfg

```
[paramiko_connection]
host_key_auto_add = True
```

Error: "No authentication methods available"

For example:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  creating new control socket for host␣
↪veos01:None as user admin
2017-04-04 12:19:05,670 p=18591 u=fred |  control socket path is /home/fred/.ansible/pc/
↪ca5960d27a
2017-04-04 12:19:05,670 p=18591 u=fred |  current working directory is /home/fred/git/
↪ansible-inc/ansible-workspace-2/test/integration
2017-04-04 12:19:05,670 p=18591 u=fred |  using connection plugin network_cli
2017-04-04 12:19:06,606 p=18591 u=fred |  connecting to host veos01 returned an error
2017-04-04 12:19:06,606 p=18591 u=fred |  No authentication methods available
2017-04-04 12:19:35,708 p=18591 u=fred |  connect retry timeout expired, unable to␣
↪connect to control socket
2017-04-04 12:19:35,709 p=18591 u=fred |  persistent_connect_retry_timeout is 15 secs
```

Suggestions to resolve:

No password or SSH key supplied

Clearing Out Persistent Connections

Platforms: Any

In Ansible 2.3, persistent connection sockets are stored in `~/.ansible/pc` for all network devices. When an Ansible playbook runs, the persistent socket connection is displayed when verbose output is specified.

`<switch> socket_path:  /home/fred/.ansible/pc/f64ddfa760`

To clear out a persistent connection before it times out (the default timeout is 30 seconds of inactivity), simple delete the socket file.

Timeout issues

Persistent connection idle timeout

By default, `ANSIBLE_PERSISTENT_CONNECT_TIMEOUT` is set to 30 (seconds). You may see the following error if this value is too low:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  persistent connection idle timeout triggered,␣
↪timeout value is 30 secs
```

Suggestions to resolve:

Increase value of persistent connection idle timeout:

```
export ANSIBLE_PERSISTENT_CONNECT_TIMEOUT=60
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
connect_timeout = 60
```

Command timeout

By default, `ANSIBLE_PERSISTENT_COMMAND_TIMEOUT` is set to 30 (seconds). Prior versions of Ansible had this value set to 10 seconds by default. You may see the following error if this value is too low:

```
2017-04-04 12:19:05,670 p=18591 u=fred |  command timeout triggered, timeout value is 30␣
↪secs
```

Suggestions to resolve:

- Option 1 (Global command timeout setting): Increase value of command timeout in configuration file or by setting environment variable.

  ```
  export ANSIBLE_PERSISTENT_COMMAND_TIMEOUT=60
  ```

  To make this a permanent change, add the following to your `ansible.cfg` file:

  ```
  [persistent_connection]
  command_timeout = 60
  ```

- Option 2 (Per task command timeout setting): Increase command timeout per task basis. All network modules support a timeout value that can be set on a per task basis. The timeout value controls the amount of time in seconds before the task will fail if the command has not returned.

  For local connection type:

  Suggestions to resolve:

  Some modules support a `timeout` option, which is different to the `timeout` keyword for tasks.

  ```
  - name: save running-config
    cisco.ios.ios_command:
      commands: copy running-config startup-config
      provider: "{{ cli }}"
      timeout: 30
  ```

  Suggestions to resolve:

  If the module does not support the `timeout` option directly, most networking connection plugins can enable similar functionality with the `ansible_command_timeout` variable.

  ```
  - name: save running-config
    cisco.ios.ios_command:
      commands: copy running-config startup-config
    vars:
      ansible_command_timeout: 60
  ```

Some operations take longer than the default 30 seconds to complete. One good example is saving the current running config on IOS devices to startup config. In this case, changing the timeout value from the default 30 seconds to 60 seconds will prevent the task from failing before the command completes successfully.

Persistent connection retry timeout

By default, `ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT` is set to 15 (seconds). You may see the following error if this value is too low:

```
2017-04-04 12:19:35,708 p=18591 u=fred |  connect retry timeout expired, unable to␣
↪connect to control socket
2017-04-04 12:19:35,709 p=18591 u=fred |  persistent_connect_retry_timeout is 15 secs
```

Suggestions to resolve:

Increase the value of the persistent connection idle timeout. Note: This value should be greater than the SSH timeout value (the timeout value under the defaults section in the configuration file) and less than the value of the persistent connection idle timeout (connect_timeout).

```
export ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT=30
```

To make this a permanent change, add the following to your `ansible.cfg` file:

```
[persistent_connection]
connect_retry_timeout = 30
```

Timeout issue due to platform specific login menu with `network_cli` connection type

In Ansible 2.9 and later, the network_cli connection plugin configuration options are added to handle the platform specific login menu. These options can be set as group/host or tasks variables.

Example: Handle single login menu prompts with host variables

```
$cat host_vars/<hostname>.yaml
---
ansible_terminal_initial_prompt:
  - "Connect to a host"
ansible_terminal_initial_answer:
  - "3"
```

Example: Handle remote host multiple login menu prompts with host variables

```
$cat host_vars/<inventory-hostname>.yaml
---
ansible_terminal_initial_prompt:
  - "Press any key to enter main menu"
  - "Connect to a host"
ansible_terminal_initial_answer:
  - "\\r"
  - "3"
ansible_terminal_initial_prompt_checkall: True
```

To handle multiple login menu prompts:

- The values of `ansible_terminal_initial_prompt` and `ansible_terminal_initial_answer` should be a list.
- The prompt sequence should match the answer sequence.
- The value of `ansible_terminal_initial_prompt_checkall` should be set to `True`.

Note: If all the prompts in sequence are not received from remote host at the time connection initialization it will result in a timeout.

Playbook issues

This section details issues are caused by issues with the Playbook itself.

Error: "Unable to enter configuration mode"

Platforms: Arista EOS and Cisco IOS

This occurs when you attempt to run a task that requires privileged mode in a user mode shell.

For example:

```
TASK [ios_system : configure name_servers]␣
↪*****************************************************************************
task path:
fatal: [ios-csr1000v]: FAILED! => {
    "changed": false,
    "failed": true,
   "msg": "unable to enter configuration mode",
}
```

Suggestions to resolve:

Use `connection: ansible.netcommon.network_cli` and `become: true`

Proxy Issues

delegate_to vs ProxyCommand

In order to use a bastion or intermediate jump host to connect to network devices over `cli` transport, network modules support the use of `ProxyCommand`.

To use `ProxyCommand`, configure the proxy settings in the Ansible inventory file to specify the proxy host.

```
[nxos]
nxos01
nxos02

[nxos:vars]
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

With the configuration above, simply build and run the playbook as normal with no additional changes necessary. The network module will now connect to the network device by first connecting to the host specified in `ansible_ssh_common_args`, which is `bastion01` in the above example.

You can also set the proxy target for all hosts by using environment variables.

```
export ANSIBLE_SSH_ARGS='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Using bastion/jump host with netconf connection

Enabling jump host setting

Bastion/jump host with netconf connection can be enabled by:

- Setting Ansible variable `ansible_netconf_ssh_config` either to `True` or custom ssh config file path
- Setting environment variable `ANSIBLE_NETCONF_SSH_CONFIG` to `True` or custom ssh config file path
- Setting `ssh_config = 1` or `ssh_config = <ssh-file-path>` under `netconf_connection` section

If the configuration variable is set to 1 the proxycommand and other ssh variables are read from default ssh config file (~/.ssh/config).

If the configuration variable is set to file path the proxycommand and other ssh variables are read from the given custom ssh file path

Example ssh config file (~/.ssh/config)

```
Host jumphost
  HostName jumphost.domain.name.com
  User jumphost-user
  IdentityFile "/path/to/ssh-key.pem"
  Port 22

# Note: Due to the way that Paramiko reads the SSH Config file,
# you need to specify the NETCONF port that the host uses.
# In other words, it does not automatically use ansible_port
# As a result you need either:

Host junos01
  HostName junos01
  ProxyCommand ssh -W %h:22 jumphost

# OR

Host junos01
  HostName junos01
  ProxyCommand ssh -W %h:830 jumphost

# Depending on the netconf port used.
```

Example Ansible inventory file

```
[junos]
junos01

[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
ansible_user=myuser
ansible_password=!vault...
```

Note: Using `ProxyCommand` with passwords through variables

By design, SSH doesn't support providing passwords through environment variables. This is done to prevent secrets from leaking out, for example in `ps` output.

We recommend using SSH Keys, and if needed an ssh-agent, rather than passwords, where ever possible.

Miscellaneous Issues

Intermittent failure while using `ansible.netcommon.network_cli` connection type

If the command prompt received in response is not matched correctly within the `ansible.netcommon.network_cli` connection plugin the task might fail intermittently with truncated response or with the error message `operation requires privilege escalation`. Starting in 2.7.1 a new buffer read timer is added to ensure prompts are matched properly and a complete response is send in output. The timer default value is 0.2 seconds and can be adjusted on a per task basis or can be set globally in seconds.

Example Per task timer setting

```
- name: gather ios facts
  cisco.ios.ios_facts:
    gather_subset: all
  register: result
  vars:
    ansible_buffer_read_timeout: 2
```

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
buffer_read_timeout = 2
```

This timer delay per command executed on remote host can be disabled by setting the value to zero.

Task failure due to mismatched error regex within command response using `ansible.netcommon.network_cli` connection type

In Ansible 2.9 and later, the `ansible.netcommon.network_cli` connection plugin configuration options are added to handle the stdout and stderr regex to identify if the command execution response consist of a normal response or an error response. These options can be set group/host variables or as tasks variables.

Example: For mismatched error response

```
- name: fetch logs from remote host
  cisco.ios.ios_command:
    commands:
      - show logging
```

Playbook run output:

```
TASK [first fetch logs] ********************************************************
fatal: [ios01]: FAILED! => {
    "changed": false,
    "msg": "RF Name:\r\n\r\n <--nsip-->
           \"IPSEC-3-REPLAY_ERROR: Test log\"\r\n*Aug  1 08:36:18.483: %SYS-7-USERLOG_
↪DEBUG:
            Message from tty578(user id: ansible): test\r\nan-ios-02#"}
```

Suggestions to resolve:

Modify the error regex for individual task.

```
- name: fetch logs from remote host
  cisco.ios.ios_command:
    commands:
      - show logging
  vars:
    ansible_terminal_stderr_re:
      - pattern: 'connection timed out'
        flags: 're.I'
```

The terminal plugin regex options `ansible_terminal_stderr_re` and `ansible_terminal_stdout_re` have `pattern` and `flags` as keys. The value of the `flags` key should be a value that is accepted by the `re.compile` python method.

Intermittent failure while using `ansible.netcommon.network_cli` connection type due to slower network or remote target host

In Ansible 2.9 and later, the `ansible.netcommon.network_cli` connection plugin configuration option is added to control the number of attempts to connect to a remote host. The default number of attempts is three. After every retry attempt the delay between retries is increased by power of 2 in seconds until either the maximum attempts are exhausted or either the `persistent_command_timeout` or `persistent_connect_timeout` timers are triggered.

To make this a global setting, add the following to your `ansible.cfg` file:

```
[persistent_connection]
network_cli_retries = 5
```

1.22.6 Working with command output and prompts in network modules

Conditionals in networking modules

Ansible allows you to use conditionals to control the flow of your playbooks. Ansible networking command modules use the following unique conditional statements.

- `eq` - Equal
- `neq` - Not equal
- `gt` - Greater than
- `ge` - Greater than or equal
- `lt` - Less than
- `le` - Less than or equal
- `contains` - Object contains specified item

Conditional statements evaluate the results from the commands that are executed remotely on the device. Once the task executes the command set, the `wait_for` argument can be used to evaluate the results before returning control to the Ansible playbook.

For example:

```
---
- name: wait for interface to be admin enabled
  arista.eos.eos_command:
      commands:
          - show interface Ethernet4 | json
      wait_for:
          - "result[0].interfaces.Ethernet4.interfaceStatus eq connected"
```

In the above example task, the command `show interface Ethernet4 | json` is executed on the remote device and the results are evaluated. If the path `(result[0].interfaces.Ethernet4.interfaceStatus)` is not equal to "connected", then the command is retried. This process continues until either the condition is satisfied or the number of retries has expired (by default, this is 10 retries at 1 second intervals).

The commands module can also evaluate more than one set of command results in an interface. For instance:

```
---
- name: wait for interfaces to be admin enabled
  arista.eos.eos_command:
      commands:
          - show interface Ethernet4 | json
          - show interface Ethernet5 | json
      wait_for:
          - "result[0].interfaces.Ethernet4.interfaceStatus eq connected"
          - "result[1].interfaces.Ethernet5.interfaceStatus eq connected"
```

In the above example, two commands are executed on the remote device, and the results are evaluated. By specifying the result index value (0 or 1), the correct result output is checked against the conditional.

The `wait_for` argument must always start with result and then the command index in `[]`, where `0` is the first command in the commands list, `1` is the second command, `2` is the third and so on.

Handling prompts in network modules

Network devices may require that you answer a prompt before performing a change on the device. Individual network modules such as cisco.ios.ios_command and cisco.nxos.nxos_command can handle this with a `prompt` parameter.

Note: `prompt` is a Python regex. If you add special characters such as `?` in the `prompt` value, the prompt won't match and you will get a timeout. To avoid this, ensure that the `prompt` value is a Python regex that matches the actual device prompt. Any special characters must be handled correctly in the `prompt` regex.

You can also use the ansible.netcommon.cli_command to handle multiple prompts.

```
---
- name: multiple prompt, multiple answer (mandatory check for all prompts)
  ansible.netcommon.cli_command:
    command: "copy sftp sftp://user@host//user/test.img"
    check_all: True
    prompt:
      - "Confirm download operation"
      - "Password"
      - "Do you want to change that to the standby image"
    answer:
      - 'y'
      - <password>
      - 'y'
```

You must list the prompt and the answers in the same order (that is, prompt[0] is answered by answer[0]).

In the above example, `check_all:  True` ensures that the task gives the matching answer to each prompt. Without that setting, a task with multiple prompts would give the first answer to every prompt.

In the following example, the second answer would be ignored and `y` would be the answer given to both prompts. That is, this task only works because both answers are identical. Also notice again that `prompt` must be a Python regex, which is why the `?` is escaped in the first prompt.

```
---
 - name: reboot ios device
   ansible.netcommon.cli_command:
     command: reload
     prompt:
       - Save\?
       - confirm
     answer:
       - y
       - y
```

See also:

Rebooting network devices with Ansible
    Examples using `wait_for`, `wait_for_connection`, and `prompt` for network devices.

Deep dive on cli_command
Detailed overview of how to use the `cli_command`.

1.22.7 Ansible Network FAQ

Topics

How can I improve performance for network playbooks?

Consider `strategy: free` if you are running on multiple hosts

The `strategy` plugin tells Ansible how to order multiple tasks on multiple hosts. *Strategy* is set at the playbook level.

The default strategy is `linear`. With strategy set to `linear`, Ansible waits until the current task has run on all hosts before starting the next task on any host. Ansible may have forks free, but will not use them until all hosts have completed the current task. If each task in your playbook must succeed on all hosts before you run the next task, use the `linear` strategy.

Using the `free` strategy, Ansible uses available forks to execute tasks on each host as quickly as possible. Even if an earlier task is still running on one host, Ansible executes later tasks on other hosts. The `free` strategy uses available forks more efficiently. If your playbook stalls on each task, waiting for one slow host, consider using `strategy: free` to boost overall performance.

Execute `show running` only if you absolutely must

The `show running` command is the most resource-intensive command to execute on a network device, because of the way queries are handled by the network OS. Using the command in your Ansible playbook will slow performance significantly, especially on large devices; repeating it will multiply the performance hit. If you have a playbook that checks the running config, then executes changes, then checks the running config again, you should expect that playbook to be very slow.

Use ProxyCommand only if you absolutely must

Network modules support the use of a *proxy or jump host* with the ProxyCommand parameter. However, when you use a jump host, Ansible must open a new SSH connection for every task, even if you are using a persistent connection type (network_cli or netconf). To maximize the performance benefits of the persistent connection types introduced in version 2.5, avoid using jump hosts whenever possible.

Set --forks to match your needs

Every time Ansible runs a task, it forks its own process. The --forks parameter defines the number of concurrent tasks - if you retain the default setting, which is --forks=5, and you are running a playbook on 10 hosts, five of those hosts will have to wait until a fork is available. Of course, the more forks you allow, the more memory and processing power Ansible will use. Since most network tasks are run on the control host, this means your laptop can quickly become cpu- or memory-bound.

Why is my output sometimes replaced with ********?

Ansible replaces any string marked no_log, including passwords, with ******** in Ansible output. This is done by design, to protect your sensitive data. Most users are happy to have their passwords redacted. However, Ansible replaces every string that matches your password with ********. If you use a common word for your password, this can be a problem. For example, if you choose Admin as your password, Ansible will replace every instance of the word Admin with ******** in your output. This may make your output harder to read. To avoid this problem, select a secure password that will not occur elsewhere in your Ansible output.

Why do the *_config modules always return changed=true with abbreviated commands?

When you issue commands directly on a network device, you can use abbreviated commands. For example, int g1/0/11 and interface GigabitEthernet1/0/11 do the same thing; shut and shutdown do the same thing. Ansible Network *_command modules work with abbreviations, because they run commands through the network OS.

When committing configuration, however, the network OS converts abbreviations into long-form commands. Whether you use shut or shutdown on GigabitEthernet1/0/11, the result in the configuration is the same: shutdown.

Ansible Network *_config modules compare the text of the commands you specify in lines to the text in the configuration. If you use shut in the lines section of your task, and the configuration reads shutdown, the module returns changed=true even though the configuration is already correct. Your task will update the configuration every time it runs.

To avoid this problem, use long-form commands with the *_config modules:

```yaml
---
- hosts: all
  gather_facts: no
  tasks:
    - cisco.ios.ios_config:
        lines:
          - shutdown
        parents: interface GigabitEthernet1/0/11
```

1.22.8 Platform Options

Some Ansible Network platforms support multiple connection types, privilege escalation (`enable` mode), or other options. The pages in this section offer standardized guides to understanding available options on each network platform. We welcome contributions from community-maintained platforms to this section.

CloudEngine OS Platform Options

CloudEngine CE OS is part of the community.network collection and supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Connections available

	CLI	NETCONF
Protocol	SSH	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	via a bastion (jump host)	via a bastion (jump host)
Connection Settings	**`ansible_connection:`** `ansible.netcommon.network_cli`	**`ansible_connection:`** `ansible.netcommon.netconf`
Enable Mode (Privilege Escalation)	not supported by ce OS	not supported by ce OS
Returned Data Format	Refer to individual module documentation	Refer to individual module documentation

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.netconf` or `ansible_connection=ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI inventory `[ce:vars]`

```
[ce:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=community.network.ce
ansible_user=myuser
ansible_password=!vault...
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords via environment variables.

Example CLI task

```
- name: Retrieve CE OS version
  community.network.ce_command:
    commands: display version
  when: ansible_network_os == 'community.network.ce'
```

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the CloudEngine OS device(s)

To enable NETCONF on a new switch using Ansible, use the `community.network.ce_config` module with the CLI connection. Set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: ansible.netcommon.network_cli
  community.network.ce_config:
    lines:
      - snetconf server enable
  when: ansible_network_os == 'community.network.ce'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory [ce:vars]

```
[ce:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=community.network.ce
ansible_user=myuser
ansible_password=!vault |
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Example NETCONF task

```
- name: Create a vlan, id is 50(ce)
  community.network.ce_vlan:
    vlan_id: 50
    name: WEB
  when: ansible_network_os == 'community.network.ce'
```

Notes

Modules that work with ansible.netcommon.network_cli

```
community.network.ce_acl_interface
community.network.ce_command
community.network.ce_config
community.network.ce_evpn_bgp
community.network.ce_evpn_bgp_rr
community.network.ce_evpn_global
community.network.ce_facts
community.network.ce_mlag_interface
community.network.ce_mtu
community.network.ce_netstream_aging
community.network.ce_netstream_export
community.network.ce_netstream_global
community.network.ce_netstream_template
community.network.ce_ntp_auth
community.network.ce_rollback
community.network.ce_snmp_contact
community.network.ce_snmp_location
community.network.ce_snmp_traps
community.network.ce_startup
community.network.ce_stp
community.network.ce_vxlan_arp
community.network.ce_vxlan_gateway
community.network.ce_vxlan_global
```

Modules that work with `ansible.netcommon.netconf`

```
community.network.ce_aaa_server
community.network.ce_aaa_server_host
community.network.ce_acl
community.network.ce_acl_advance
community.network.ce_bfd_global
community.network.ce_bfd_session
community.network.ce_bfd_view
community.network.ce_bgp
community.network.ce_bgp_af
community.network.ce_bgp_neighbor
community.network.ce_bgp_neighbor_af
community.network.ce_dldp
community.network.ce_dldp_interface
community.network.ce_eth_trunk
community.network.ce_evpn_bd_vni
community.network.ce_file_copy
community.network.ce_info_center_debug
community.network.ce_info_center_global
community.network.ce_info_center_log
community.network.ce_info_center_trap
community.network.ce_interface
community.network.ce_interface_ospf
community.network.ce_ip_interface
community.network.ce_lacp
community.network.ce_link_status
community.network.ce_lldp
community.network.ce_lldp_interface
community.network.ce_mlag_config
community.network.ce_netconf
community.network.ce_ntp
community.network.ce_ospf
community.network.ce_ospf_vrf
community.network.ce_reboot
community.network.ce_sflow
community.network.ce_snmp_community
community.network.ce_snmp_target_host
community.network.ce_snmp_user
community.network.ce_static_route
community.network.ce_static_route_bfd
community.network.ce_switchport
community.network.ce_vlan
community.network.ce_vrf
community.network.ce_vrf_af
community.network.ce_vrf_interface
community.network.ce_vrrp
community.network.ce_vxlan_tunnel
community.network.ce_vxlan_vap
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

CNOS Platform Options

CNOS is part of the community.network collection and supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on CNOS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/cnos.yml`

```yaml
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.cnos
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.

- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Retrieve CNOS OS version
  community.network.cnos_command:
    commands: show version
  when: ansible_network_os == 'community.network.cnos'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Dell OS6 Platform Options

The dellemc.os6 collection supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on OS6 in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	through a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/dellos6.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: dellemc.os6.os6
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (dellos6)
  dellemc.os6.os6_config:
    backup: yes
  register: backup_dellso6_location
  when: ansible_network_os == 'dellemc.os6.os6'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Dell OS9 Platform Options

The dellemc.os9 collection supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on OS9 in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	through a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/dellos9.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: dellemc.os9.os9
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (dellos9)
  dellemc.os9.os9_config:
    backup: yes
  register: backup_dellos9_location
  when: ansible_network_os == 'dellemc.os9.os9'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Dell OS10 Platform Options

The dellemc.os10 collection supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on OS10 in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	through a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/dellos10.yml`

```yaml
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: dellemc.os10.os10
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.

- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the ProxyCommand directive. To prevent secrets from leaking out (for example in ps output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (dellos10)
  dellemc.os10.os10_config:
    backup: yes
  register: backup_dellos10_location
  when: ansible_network_os == 'dellemc.os10.os10'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

ENOS Platform Options

ENOS is part of the community.network collection and supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on ENOS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The ansible_connection: local has been deprecated. Please use ansible_connection: ansible.netcommon.network_cli instead.

Using CLI in Ansible

Example CLI group_vars/enos.yml

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.enos
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the ansible_password configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the ansible_ssh_common_args configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the ProxyCommand directive. To prevent secrets from leaking out (for example in ps output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Retrieve ENOS OS version
  community.network.enos_command:
    commands: show version
  when: ansible_network_os == 'community.network.enos'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

EOS Platform Options

The Arista EOS collection supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Connections available

	CLI	eAPI
Protocol	SSH	HTTP(S)
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password	uses HTTPS certificates if present
Indirect Access	by a bastion (jump host)	through a web proxy
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`	`ansible_connection: ansible.netcommon.httpapi`
Enable Mode (Privilege Escalation)	supported: • use `ansible_become: true` with `ansible_become_method: enable`	supported: • `httpapi` uses `ansible_become: true` with `ansible_become_method: enable`
Returned Data Format	`stdout[0].`	`stdout[0].messages[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` or `ansible_connection: ansible.netcommon.httpapi` instead.

Using CLI in Ansible

Example CLI `group_vars/eos.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: arista.eos.eos
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.

- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the ProxyCommand directive. To prevent secrets from leaking out (for example in ps output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (eos)
  arista.eos.eos_config:
    backup: yes
  register: backup_eos_location
  when: ansible_network_os == 'arista.eos.eos'
```

Using eAPI in Ansible

Enabling eAPI

Before you can use eAPI to connect to a switch, you must enable eAPI. To enable eAPI on a new switch with Ansible, use the `arista.eos.eos_eapi` module through the CLI connection. Set up `group_vars/eos.yml` just like in the CLI example above, then run a playbook task like this:

```
- name: Enable eAPI
  arista.eos.eos_eapi:
    enable_http: yes
    enable_https: yes
  become: true
  become_method: enable
  when: ansible_network_os == 'arista.eos.eos'
```

You can find more options for enabling HTTP/HTTPS connections in the arista.eos.eos_eapi module documentation.

Once eAPI is enabled, change your `group_vars/eos.yml` to use the eAPI connection.

Example eAPI `group_vars/eos.yml`

```
ansible_connection: ansible.netcommon.httpapi
ansible_network_os: arista.eos.eos
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
proxy_env:
  http_proxy: http://proxy.example.com:8080
```

- If you are accessing your host directly (not through a web proxy) you can remove the `proxy_env` configuration.
- If you are accessing your host through a web proxy using `https`, change `http_proxy` to `https_proxy`.

Example eAPI task

```
- name: Backup current switch config (eos)
  arista.eos.eos_config:
    backup: yes
  register: backup_eos_location
  environment: "{{ proxy_env }}"
  when: ansible_network_os == 'arista.eos.eos'
```

In this example the `proxy_env` variable defined in `group_vars` gets passed to the `environment` option of the module in the task.

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

ERIC_ECCLI Platform Options

Extreme ERIC_ECCLI is part of the community.network collection and only supports CLI connections today. This page offers details on how to use `ansible.netcommon.network_cli` on ERIC_ECCLI in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported by ERIC_ECCLI
Returned Data Format	`stdout[0].`

ERIC_ECCLI does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI group_vars/eric_eccli.yml

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.eric_eccli
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: run show version on remote devices (eric_eccli)
  community.network.eric_eccli_command:
     commands: show version
  when: ansible_network_os == 'community.network.eric_eccli'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

EXOS Platform Options

Extreme EXOS is part of the community.network collection and supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Connections available

	CLI	EXOS-API
Protocol	SSH	HTTP(S)
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password	uses HTTPS certificates if present
Indirect Access	by a bastion (jump host)	through a web proxy
Connection Settings	**`ansible_connection:`** `ansible.netcommon.network_cli`	**`ansible_connection:`** `ansible.netcommon.httpapi`
Enable Mode (Privilege Escalation)	not supported by EXOS	not supported by EXOS
Returned Data Format	`stdout[0].`	`stdout[0].messages[0].`

EXOS does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli` or `ansible_connection: ansible.netcommon.httpapi`.

Using CLI in Ansible

Example CLI `group_vars/exos.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.exos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Retrieve EXOS OS version
  community.network.exos_command:
    commands: show version
  when: ansible_network_os == 'community.network.exos'
```

Using EXOS-API in Ansible

Example EXOS-API `group_vars/exos.yml`

```yaml
ansible_connection: ansible.netcommon.httpapi
ansible_network_os: community.network.exos
ansible_user: myuser
ansible_password: !vault...
proxy_env:
  http_proxy: http://proxy.example.com:8080
```

- If you are accessing your host directly (not through a web proxy) you can remove the `proxy_env` configuration.
- If you are accessing your host through a web proxy using `https`, change `http_proxy` to `https_proxy`.

Example EXOS-API task

```yaml
- name: Retrieve EXOS OS version
  community.network.exos_command:
    commands: show version
  when: ansible_network_os == 'community.network.exos'
```

In this example the `proxy_env` variable defined in `group_vars` gets passed to the `environment` option of the module used in the task.

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

FRR Platform Options

The FRR collection supports the `ansible.netcommon.network_cli` connection. This section provides details on how to use this connection for Free Range Routing (FRR).

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported
Returned Data Format	`stdout[0].`

Using CLI in Ansible

Example CLI `group_vars/frr.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: frr.frr.frr
ansible_user: frruser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- The `ansible_user` should be a part of the `frrvty` group and should have the default shell set to `/bin/vtysh`.
- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Gather FRR facts
  frr.frr.frr_facts:
    gather_subset:
     - config
     - hardware
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

ICX Platform Options

ICX is part of the community.network collection supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on ICX in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

Using CLI in Ansible

Example CLI `group_vars/icx.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.icx
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (icx)
  community.network.icx_config:
    backup: yes
  register: backup_icx_location
  when: ansible_network_os == 'community.network.icx'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

IOS Platform Options

The Cisco IOS collection supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on IOS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/ios.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.ios.ios
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (ios)
  cisco.ios.ios_config:
    backup: yes
  register: backup_ios_location
  when: ansible_network_os == 'cisco.ios.ios'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

IOS-XR Platform Options

The Cisco IOS-XR collection supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Connections available

	CLI	NETCONF only for modules `iosxr_banner`, `iosxr_interface`, `iosxr_logging`, `iosxr_system`, `iosxr_user`
Protocol	SSH	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)	by a bastion (jump host)
Connection Settings	**ansible_connection:** ansible.netcommon.network_cli	**ansible_connection:** ansible.netcommon.netconf
Enable Mode (Privilege Escalation)	not supported	not supported
Returned Data Format	Refer to individual module documentation	Refer to individual module documentation

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` or `ansible_connection: ansible.netcommon.netconf` instead.

Using CLI in Ansible

Example CLI inventory `[iosxr:vars]`

```
[iosxr:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=cisco.iosxr.iosxr
ansible_user=myuser
ansible_password=!vault...
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Retrieve IOS-XR version
  cisco.iosxr.iosxr_command:
    commands: show version
  when: ansible_network_os == 'cisco.iosxr.iosxr'
```

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the Cisco IOS-XR device(s)

To enable NETCONF on a new switch via Ansible, use the `cisco.iosxr.iosxr_netconf` module through the CLI connection. Set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: ansible.netcommon.network_cli
  cisco.iosxr.iosxr_netconf:
  when: ansible_network_os == 'cisco.iosxr.iosxr'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory `[iosxr:vars]`

```
[iosxr:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=cisco.iosxr.iosxr
ansible_user=myuser
ansible_password=!vault |
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Example NETCONF task

```
- name: Configure hostname and domain-name
  cisco.iosxr.iosxr_system:
    hostname: iosxr01
    domain_name: test.example.com
    domain_search:
      - ansible.com
      - redhat.com
      - cisco.com
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

IronWare Platform Options

IronWare is part of the community.network collection and supports Enable Mode (Privilege Escalation). This page offers details on how to use Enable Mode on IronWare in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`
Returned Data Format	`stdout[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/mlx.yml`

```yaml
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.ironware
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (ironware)
  community.network.ironware_config:
    backup: yes
  register: backup_ironware_location
  when: ansible_network_os == 'community.network.ironware'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Junos OS Platform Options

The Juniper Junos OS supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Connections available

	CLI `junos_netconf` & `junos_command` modules only	NETCONF all modules except `junos_netconf`, which enables NETCONF
Protocol	SSH	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)	by a bastion (jump host)
Connection Settings	`ansible_connection: ``ansible.netcommon.network_cli`	`ansible_connection: ``ansible.netcommon.netconf`
Enable Mode (Privilege Escalation)	not supported by Junos OS	not supported by Junos OS
Returned Data Format	`stdout[0].`	• json: `result[0]['software-information'][0]['` `foo lo0` • text: `result[1].interface-information[0].physical-interface[0].name[0].data foo lo0` • xml: `result[1].rpc-reply.interface-information[0].physical-interface[0].name[0].data foo lo0`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` or `ansible_connection: ansible.netcommon.netconf` instead.

Using CLI in Ansible

Example CLI inventory `[junos:vars]`

```
[junos:vars]
ansible_connection=ansible.netcommon.network_cli
ansible_network_os=junipernetworks.junos.junos
ansible_user=myuser
ansible_password=!vault...
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support

providing passwords through environment variables.

Example CLI task

```
- name: Retrieve Junos OS version
  junipernetworks.junos.junos_command:
    commands: show version
  when: ansible_network_os == 'junipernetworks.junos.junos'
```

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the Junos OS device(s)

To enable NETCONF on a new switch through Ansible, use the `junipernetworks.junos.junos_netconf` module through the CLI connection. Set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: ansible.netcommon.network_cli
  junipernetworks.junos.junos_netconf:
  when: ansible_network_os == 'junipernetworks.junos.junos'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory `[junos:vars]`

```
[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
ansible_user=myuser
ansible_password=!vault |
ansible_ssh_common_args='-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

Example NETCONF task

```
- name: Backup current switch config (junos)
  junipernetworks.junos.junos_config:
    backup: yes
  register: backup_junos_location
  when: ansible_network_os == 'junipernetworks.junos.junos'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Meraki Platform Options

The cisco.meraki collection only supports the `local` connection type at this time.

Connections available

	Dashboard API
Protocol	HTTP(S)
Credentials	uses API key from Dashboard
Connection Settings	`ansible_connection: localhost`
Returned Data Format	`data.`

Example Meraki task

```yaml
cisco.meraki.meraki_organization:
  auth_key: abc12345
  org_name: YourOrg
  state: present
delegate_to: localhost
```

See also:

Setting timeout options

Pluribus NETVISOR Platform Options

Pluribus NETVISOR Ansible is part of the community.network collection and only supports CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `ansible.netcommon.network_cli` on NETVISOR in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported by NETVISOR
Returned Data Format	`stdout[0].`

Pluribus NETVISOR does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI `group_vars/netvisor.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.netcommon.netvisor
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Create access list
  community.network.pn_access_list:
    pn_name: "foo"
    pn_scope: "local"
    state: "present"
  register: acc_list
  when: ansible_network_os == 'community.network.netvisor'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

NOS Platform Options

Extreme NOS is part of the community.network collection and only supports CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `ansible.netcommon.network_cli` on NOS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: community.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported by NOS
Returned Data Format	`stdout[0].`

NOS does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI `group_vars/nos.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.nos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.

- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Get version information (nos)
  community.network.nos_command:
    commands: "show version"
  register: show_ver
  when: ansible_network_os == 'community.network.nos'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

NXOS Platform Options

The Cisco NXOS supports multiple connections. This page offers details on how each connection works in Ansible and how to use it.

Connections available

	CLI	NX-API
Protocol	SSH	HTTP(S)
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password	uses HTTPS certificates if present
Indirect Access	by a bastion (jump host)	by a web proxy
Connection Settings	**`ansible_connection:`** `ansible.netcommon.network_cli`	**`ansible_connection:`** `ansible.netcommon.httpapi`
Enable Mode (Privilege Escalation) supported as of 2.5.3	supported: use `ansible_become: true` with `ansible_become_method: enable` and `ansible_become_password:`	not supported by NX-API
Returned Data Format	`stdout[0].`	`stdout[0].messages[0].`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` or `ansible_connection: ansible.netcommon.httpapi` instead.

Using CLI in Ansible

Example CLI `group_vars/nxos.yml`

```yaml
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.nxos.nxos
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (nxos)
  cisco.nxos.nxos_config:
    backup: yes
  register: backup_nxos_location
  when: ansible_network_os == 'cisco.nxos.nxos'
```

Using NX-API in Ansible

Enabling NX-API

Before you can use NX-API to connect to a switch, you must enable NX-API. To enable NX-API on a new switch through Ansible, use the `nxos_nxapi` module through the CLI connection. Set up group_vars/nxos.yml just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NX-API
   cisco.nxos.nxos_nxapi:
       enable_http: yes
       enable_https: yes
   when: ansible_network_os == 'cisco.nxos.nxos'
```

To find out more about the options for enabling HTTP/HTTPS and local http see the nxos_nxapi module documentation.

Once NX-API is enabled, change your `group_vars/nxos.yml` to use the NX-API connection.

Example NX-API `group_vars/nxos.yml`

```
ansible_connection: ansible.netcommon.httpapi
ansible_network_os: cisco.nxos.nxos
ansible_user: myuser
ansible_password: !vault...
proxy_env:
  http_proxy: http://proxy.example.com:8080
```

- If you are accessing your host directly (not through a web proxy) you can remove the `proxy_env` configuration.
- If you are accessing your host through a web proxy using `https`, change `http_proxy` to `https_proxy`.

Example NX-API task

```
- name: Backup current switch config (nxos)
  cisco.nxos.nxos_config:
    backup: yes
  register: backup_nxos_location
  environment: "{{ proxy_env }}"
  when: ansible_network_os == 'cisco.nxos.nxos'
```

In this example the `proxy_env` variable defined in `group_vars` gets passed to the `environment` option of the module used in the task.

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

Cisco Nexus platform support matrix

The following platforms and software versions have been certified by Cisco to work with this version of Ansible.

Table 4: Platform / Software Minimum Requirements

Supported Platforms	Minimum NX-OS Version
Cisco Nexus N3k	7.0(3)I2(5) and later
Cisco Nexus N9k	7.0(3)I2(5) and later
Cisco Nexus N5k	7.3(0)N1(1) and later
Cisco Nexus N6k	7.3(0)N1(1) and later
Cisco Nexus N7k	7.3(0)D1(1) and later
Cisco Nexus MDS	8.4(1) and later (Please see individual module documentation for compatibility)

Table 5: Platform Models

Platform	Description
N3k	Support includes N30xx, N31xx and N35xx models
N5k	Support includes all N5xxx models
N6k	Support includes all N6xxx models
N7k	Support includes all N7xxx models
N9k	Support includes all N9xxx models
MDS	Support includes all MDS 9xxx models

See also:

Setting timeout options

RouterOS Platform Options

RouterOS is part of the community.network collection and only supports CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `ansible.netcommon.network_cli` on RouterOS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.network.network_cli`
Enable Mode (Privilege Escalation)	not supported by RouterOS
Returned Data Format	`stdout[0].`

RouterOS does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI `group_vars/routeros.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.routeros
ansible_user: myuser
ansible_password: !vault...
ansible_become: true
ansible_become_method: enable
ansible_become_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.
- If you are getting timeout errors you may want to add `+cet1024w` suffix to your username which will disable console colors, enable “dumb” mode, tell RouterOS not to try detecting terminal capabilities and set terminal width to 1024 columns. See article Console login process in MikroTik wiki for more information.

Example CLI task

```
- name: Display resource statistics (routeros)
  community.network.routeros_command:
    commands: /system resource print
  register: routeros_resources
  when: ansible_network_os == 'community.network.routeros'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

SLX-OS Platform Options

Extreme SLX-OS is part of the community.network collection and only supports CLI connections today. `httpapi` modules may be added in future. This page offers details on how to use `ansible.netcommon.network_cli` on SLX-OS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported by SLX-OS
Returned Data Format	`stdout[0].`

SLX-OS does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI `group_vars/slxos.yml`

```yaml
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.slxos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.

- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Backup current switch config (slxos)
  community.network.slxos_config:
    backup: yes
  register: backup_slxos_location
  when: ansible_network_os == 'community.network.slxos'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

VOSS Platform Options

Extreme VOSS is part of the community.network collection and only supports CLI connections today. This page offers details on how to use `ansible.netcommon.network_cli` on VOSS in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	supported: use `ansible_become: true` with `ansible_become_method: enable`
Returned Data Format	`stdout[0].`

VOSS does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI `group_vars/voss.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.voss
ansible_user: myuser
ansible_become: true
ansible_become_method: enable
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Retrieve VOSS info
  community.network.voss_command:
    commands: show sys-info
  when: ansible_network_os == 'community.network.voss'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

VyOS Platform Options

The VyOS collection supports the `ansible.netcommon.network_cli` connection type. This page offers details on connection options to manage VyOS using Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported
Returned Data Format	Refer to individual module documentation

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.network_cli` instead.

Using CLI in Ansible

Example CLI `group_vars/vyos.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: vyos.vyos.vyos
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.
- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Retrieve VyOS version info
  vyos.vyos.vyos_command:
    commands: show version
  when: ansible_network_os == 'vyos.vyos.vyos'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

WeOS 4 Platform Options

Westermo WeOS 4 is part of the community.network collection and only supports CLI connections. This page offers details on how to use `ansible.netcommon.network_cli` on WeOS 4 in Ansible.

Connections available

	CLI
Protocol	SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	by a bastion (jump host)
Connection Settings	`ansible_connection: community.netcommon.network_cli`
Enable Mode (Privilege Escalation)	not supported by WeOS 4
Returned Data Format	`stdout[0].`

WeOS 4 does not support `ansible_connection: local`. You must use `ansible_connection: ansible.netcommon.network_cli`.

Using CLI in Ansible

Example CLI `group_vars/weos4.yml`

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: community.network.weos4
ansible_user: myuser
ansible_password: !vault...
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q bastion01"'
```

- If you are using SSH keys (including an ssh-agent) you can remove the `ansible_password` configuration.
- If you are accessing your host directly (not through a bastion/jump host) you can remove the `ansible_ssh_common_args` configuration.

- If you are accessing your host through a bastion/jump host, you cannot include your SSH password in the `ProxyCommand` directive. To prevent secrets from leaking out (for example in `ps` output), SSH does not support providing passwords through environment variables.

Example CLI task

```
- name: Get version information (WeOS 4)
  ansible.netcommon.cli_command:
    commands: "show version"
  register: show_ver
  when: ansible_network_os == 'community.network.weos4'
```

Example Configuration task

```
- name: Replace configuration with file on ansible host (WeOS 4)
  ansible.netcommon.cli_config:
    config: "{{ lookup('file', 'westermo.conf') }}"
    replace: "yes"
    diff_match: exact
    diff_replace: config
  when: ansible_network_os == 'community.network.weos4'
```

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Netconf enabled Platform Options

This page offers details on how the netconf connection works in Ansible and how to use it.

Connections available

	NETCONF all modules except `junos_netconf`, which enables NETCONF
Protocol	XML over SSH
Credentials	uses SSH keys / SSH-agent if present accepts `-u myuser -k` if using password
Indirect Access	through a bastion (jump host)
Connection Settings	`ansible_connection: ansible.netcommon.netconf`

The `ansible_connection: local` has been deprecated. Please use `ansible_connection: ansible.netcommon.netconf` instead.

Using NETCONF in Ansible

Enabling NETCONF

Before you can use NETCONF to connect to a switch, you must:

- install the `ncclient` Python package on your control node(s) with `pip install ncclient`
- enable NETCONF on the Junos OS device(s)

To enable NETCONF on a new switch through Ansible, use the platform specific module through the CLI connection or set it manually. For example set up your platform-level variables just like in the CLI example above, then run a playbook task like this:

```
- name: Enable NETCONF
  connection: ansible.netcommon.network_cli
  junipernetworks.junos.junos_netconf:
  when: ansible_network_os == 'junipernetworks.junos.junos'
```

Once NETCONF is enabled, change your variables to use the NETCONF connection.

Example NETCONF inventory `[junos:vars]`

```
[junos:vars]
ansible_connection=ansible.netcommon.netconf
ansible_network_os=junipernetworks.junos.junos
ansible_user=myuser
ansible_password=!vault |
```

Example NETCONF task

```
- name: Backup current switch config
  junipernetworks.junos.netconf_config:
    backup: yes
  register: backup_junos_location
```

Example NETCONF task with configurable variables

```
- name: configure interface while providing different private key file path
  junipernetworks.junos.netconf_config:
    backup: yes
  register: backup_junos_location
  vars:
    ansible_private_key_file: /home/admin/.ssh/newprivatekeyfile
```

Note: For netconf connection plugin configurable variables see ansible.netcommon.netconf.

Bastion/Jumphost configuration

To use a jump host to connect to a NETCONF enabled device you must set the `ANSIBLE_NETCONF_SSH_CONFIG` environment variable.

`ANSIBLE_NETCONF_SSH_CONFIG` can be set to either:

- 1 or TRUE (to trigger the use of the default SSH config file ~/.ssh/config)
- The absolute path to a custom SSH config file.

The SSH config file should look something like:

```
Host *
  proxycommand ssh -o StrictHostKeyChecking=no -W %h:%p jumphost-username@jumphost.fqdn.
↪com
  StrictHostKeyChecking no
```

Authentication for the jump host must use key based authentication.

You can either specify the private key used in the SSH config file:

```
IdentityFile "/absolute/path/to/private-key.pem"
```

Or you can use an ssh-agent.

ansible_network_os auto-detection

If `ansible_network_os` is not specified for a host, then Ansible will attempt to automatically detect what `network_os` plugin to use.

`ansible_network_os` auto-detection can also be triggered by using `auto` as the `ansible_network_os`. (Note: Previously `default` was used instead of `auto`).

Warning: Never store passwords in plain text. We recommend using SSH keys to authenticate SSH connections. Ansible supports ssh-agent to manage your SSH keys. If you must use passwords to authenticate SSH connections, we recommend encrypting them with *Ansible Vault*.

See also:

Setting timeout options

Settings by Platform

		`ansible_connection:` settings a	
Network OS	`ansible_network_os:`	network_cli	netco
Arista EOS *[†]*	`arista.eos.eos`	✓	
Ciena SAOS6	`ciena.saos6.saos6`	✓	
Cisco ASA *[†]*	`cisco.asa.asa`	✓	
Cisco IOS *[†]*	`cisco.ios.ios`	✓	
Cisco IOS XR *[†]*	`cisco.iosxr.iosxr`	✓	
Cisco NX-OS *[†]*	`cisco.nxos.nxos`	✓	
Cloudengine OS	`community.network.ce`	✓	✓
Dell OS6	`dellemc.os6.os6`	✓	
Dell OS9	`dellemc.os9.os9`	✓	
Dell OS10	`dellemc.os10.os10`	✓	
Ericsson ECCLI	`community.network.eric_eccli`	✓	
Extreme EXOS	`community.network.exos`	✓	
Extreme IronWare	`community.network.ironware`	✓	
Extreme NOS	`community.network.nos`	✓	
Extreme SLX-OS	`community.network.slxos`	✓	
Extreme VOSS	`community.network.voss`	✓	
F5 BIG-IP			
F5 BIG-IQ			
Junos OS *[†]*	`junipernetworks.junos.junos`	✓	✓
Lenovo CNOS	`community.network.cnos`	✓	
Lenovo ENOS	`community.network.enos`	✓	
Meraki			
MikroTik RouterOS	`community.network.routeros`	✓	
Nokia SR OS			
Pluribus Netvisor	`community.network.netvisor`	✓	
Ruckus ICX	`community.network.icx`	✓	
VyOS *[†]*	`vyos.vyos.vyos`	✓	
Westermo WeOS 4	`community.network.weos4`	✓	
OS that supports Netconf *[†]*	`<network-os>`		✓

[†] Maintained by Ansible Network Team

1.23 Network Developer Guide

Welcome to the Developer Guide for Ansible Network Automation!

Who should use this guide?

If you want to extend Ansible for Network Automation by creating a module or plugin, this guide is for you. This guide is specific to networking. You should already be familiar with how to create, test, and document modules and plugins, as well as the prerequisites for getting your module or plugin accepted into the main Ansible repository. See the *Developer Guide* for details. Before you proceed, please read:

- How to *add a custom plugin or module locally*.
- How to figure out if *developing a module is the right approach* for my use case.
- How to *set up my Python development environment*.
- How to *get started writing a module*.

Find the network developer task that best describes what you want to do:

- I want to *develop a network resource module*.
- I want to *develop a network connection plugin*.
- I want to *document my set of modules for a network platform*.

If you prefer to read the entire guide, here's a list of the pages in order.

1.23.1 Developing network resource modules

Understanding network and security resource modules

Network and security devices separate configuration into sections (such as interfaces, VLANs, and so on) that apply to a network or security service. Ansible resource modules take advantage of this to allow users to configure subsections or resources within the device configuration. Resource modules provide a consistent experience across different network and security devices. For example, a network resource module may only update the configuration for a specific portion of the network interfaces, VLANs, ACLs, and so on for a network device. The resource module:

1. Fetches a piece of the configuration (fact gathering), for example, the interfaces configuration.
2. Converts the returned configuration into key-value pairs.
3. Places those key-value pairs into an internal independent structured data format.

Now that the configuration data is normalized, the user can update and modify the data and then use the resource module to send the configuration data back to the device. This results in a full round-trip configuration update without the need for manual parsing, data manipulation, and data model management.

The resource module has two top-level keys - `config` and `state`:

- `config` defines the resource configuration data model as key-value pairs. The type of the `config` option can be `dict` or `list of dict` based on the resource managed. That is, if the device has a single global configuration, it should be a `dict` (for example, a global LLDP configuration). If the device has multiple instances of configuration, it should be of type `list` with each element in the list of type `dict` (for example, interfaces configuration).
- `state` defines the action the resource module takes on the end device.

The `state` for a new resource module should support the following values (as applicable for the devices that support them):

merged
: Ansible merges the on-device configuration with the provided configuration in the task.

replaced
: Ansible replaces the on-device configuration subsection with the provided configuration subsection in the task.

overridden
: Ansible overrides the on-device configuration for the resource with the provided configuration in the task. Use caution with this state as you could remove your access to the device (for example, by overriding the management interface configuration).

deleted
: Ansible deletes the on-device configuration subsection and restores any default settings.

gathered
: Ansible displays the resource details gathered from the network device and accessed with the `gathered` key in the result.

rendered
: Ansible renders the provided configuration in the task in the device-native format (for example, Cisco IOS CLI). Ansible returns this rendered configuration in the `rendered` key in the result. Note this state does not communicate with the network device and can be used offline.

parsed
: Ansible parses the configuration from the `running_configuration` option into Ansible structured data in the

`parsed` key in the result. Note this does not gather the configuration from the network device so this state can be used offline.

Modules in Ansible-maintained collections must support these state values. If you develop a module with only "present" and "absent" for state, you may submit it to a community collection.

Note: The states `rendered`, `gathered`, and `parsed` do not perform any change on the device.

See also:

Deep Dive on VLANs Resource Modules for Network Automation
Walkthrough of how state values are implemented for VLANs.

Developing network and security resource modules

The Ansible Engineering team ensures the module design and code pattern within Ansible-maintained collections is uniform across resources and across platforms to give a vendor-independent feel and deliver good quality code. We recommend you use the resource module builder to develop a resource module.

The highlevel process for developing a resource module is:

1. Create and share a resource model design in the resource module models repository as a PR for review.
2. Download the latest version of the resource module builder.
3. Run the `resource module builder` to create a collection scaffold from your approved resource model.
4. Write the code to implement your resource module.
5. Develop integration and unit tests to verify your resource module.
6. Create a PR to the appropriate collection that you want to add your new resource module to. See *Contributing to Ansible-maintained Collections* for details on determining the correct collection for your module.

Understanding the model and resource module builder

The resource module builder is an Ansible Playbook that helps developers scaffold and maintain an Ansible resource module. It uses a model as the single source of truth for the module. This model is a `yaml` file that is used for the module DOCUMENTATION section and the argument spec.

The resource module builder has the following capabilities:

- Uses a defined model to scaffold a resource module directory layout and initial class files.
- Scaffolds either an Ansible role or a collection.
- Subsequent uses of the resource module builder will only replace the module arspec and file containing the module docstring.
- Allows you to store complex examples along side the model in the same directory.
- Maintains the model as the source of truth for the module and use resource module builder to update the source files as needed.
- Generates working sample modules for both `<network_os>_<resource>` and `<network_os>_facts`.

Accessing the resource module builder

To access the resource module builder:

1. clone the github repository:

```
git clone https://github.com/ansible-network/resource_module_builder.git
```

2. Install the requirements:

```
pip install -r requirements.txt
```

Creating a model

You must create a model for your new resource. The model is the single source of truth for both the argspec and docstring, keeping them in sync. Once your model is approved, you can use the resource module builder to generate three items based on the model:

- The scaffold for a new module
- The argspec for the new module
- The docstring for the new module

For any subsequent changes to the functionality, update the model first and use the resource module builder to update the module argspec and docstring.

For example, the resource model builder includes the `myos_interfaces.yml` sample in the `models` directory, as seen below:

```
---
GENERATOR_VERSION: '1.0'

NETWORK_OS: myos
RESOURCE: interfaces
COPYRIGHT: Copyright 2019 Red Hat
LICENSE: gpl-3.0.txt

DOCUMENTATION: |
  module: myos_interfaces
  version_added: 1.0.0
  short_description: 'Manages <xxxx> attributes of <network_os> <resource>'
  description: 'Manages <xxxx> attributes of <network_os> <resource>.'
  author: Ansible Network Engineer
 notes:
    - 'Tested against <network_os> <version>'
  options:
    config:
      description: The provided configuration
      type: list
      elements: dict
      suboptions:
        name:
          type: str
          description: The name of the <resource>
```

(continues on next page)

(continued from previous page)

```
        some_string:
          type: str
          description:
          - The some_string_01
          choices:
          - choice_a
          - choice_b
          - choice_c
          default: choice_a
        some_bool:
          description:
          - The some_bool.
          type: bool
        some_int:
          description:
          - The some_int.
          type: int
          version_added: '1.1.0'
        some_dict:
          type: dict
          description:
          - The some_dict.
          suboptions:
            property_01:
              description:
              - The property_01
              type: str
    state:
      description:
      - The state of the configuration after module completion.
      type: str
      choices:
      - merged
      - replaced
      - overridden
      - deleted
      default: merged
EXAMPLES:
  - deleted_example_01.txt
  - merged_example_01.txt
  - overridden_example_01.txt
  - replaced_example_01.txt
```

Notice that you should include examples for each of the states that the resource supports. The resource module builder also includes these in the sample model.

Share this model as a PR for review at resource module models repository. You can also see more model examples at that location.

Creating a collection scaffold from a resource model

To use the resource module builder to create a collection scaffold from your approved resource model:

```
ansible-playbook -e rm_dest=<destination for modules and module utils> \
                 -e structure=collection \
                 -e collection_org=<collection_org> \
                 -e collection_name=<collection_name> \
                 -e model=<model> \
                 site.yml
```

Where the parameters are as follows:

- `rm_dest`: The directory where the resource module builder places the files and directories for the resource module and facts modules.
- `structure`: The directory layout type (role or collection)
  - `role`: Generate a role directory layout.
  - `collection`: Generate a collection directory layout.
- `collection_org`: The organization of the collection, required when *structure=collection*.
- `collection_name`: The name of the collection, required when *structure=collection*.
- `model`: The path to the model file.

To use the resource module builder to create a role scaffold:

```
ansible-playbook -e rm_dest=<destination for modules and module utils> \
                 -e structure=role \
                 -e model=<model> \
                 site.yml
```

Examples

Collection directory layout

This example shows the directory layout for the following:

- `network_os`: myos
- `resource`: interfaces

```
ansible-playbook -e rm_dest=~/github/rm_example \
                 -e structure=collection \
                 -e collection_org=cidrblock \
                 -e collection_name=my_collection \
                 -e model=models/myos/interfaces/myos_interfaces.yml \
                 site.yml
```

```
├── docs
├── LICENSE.txt
├── playbooks
├── plugins
|   ├── action
```

(continues on next page)

(continued from previous page)

```
|   ├── filter
|   ├── inventory
|   ├── modules
|   |   ├── __init__.py
|   |   ├── myos_facts.py
|   |   └──  myos_interfaces.py
|   └──  module_utils
|       ├── __init__.py
|       └──  network
|           ├── __init__.py
|           └──  myos
|               ├── argspec
|               |   ├── facts
|               |   |   ├── facts.py
|               |   |   └──  __init__.py
|               |   ├── __init__.py
|               |   └──  interfaces
|               |       ├── __init__.py
|               |       └──  interfaces.py
|               ├── config
|               |   ├── __init__.py
|               |   └──  interfaces
|               |       ├── __init__.py
|               |       └──  interfaces.py
|               ├── facts
|               |   ├── facts.py
|               |   ├── __init__.py
|               |   └──  interfaces
|               |       ├── __init__.py
|               |       └──  interfaces.py
|               ├── __init__.py
|               └──  utils
|                   ├── __init__.py
|                   └──  utils.py
|
├── README.md
└──  roles
```

Role directory layout

This example displays the role directory layout for the following:

- `network_os`: myos
- `resource`: interfaces

```
ansible-playbook -e rm_dest=~/github/rm_example/roles/my_role \
                 -e structure=role \
                 -e model=models/myos/interfaces/myos_interfaces.yml \
                 site.yml
```

```
roles
└── my_role
    ├── library
    │   ├── __init__.py
    │   ├── myos_facts.py
    │   └── myos_interfaces.py
    ├── LICENSE.txt
    ├── module_utils
    │   ├── __init__.py
    │   └── network
    │       ├── __init__.py
    │       └── myos
    │           ├── argspec
    │           │   ├── facts
    │           │   │   ├── facts.py
    │           │   │   └── __init__.py
    │           │   ├── __init__.py
    │           │   └── interfaces
    │           │       ├── __init__.py
    │           │       └── interfaces.py
    │           ├── config
    │           │   ├── __init__.py
    │           │   └── interfaces
    │           │       ├── __init__.py
    │           │       └── interfaces.py
    │           ├── facts
    │           │   ├── facts.py
    │           │   ├── __init__.py
    │           │   └── interfaces
    │           │       ├── __init__.py
    │           │       └── interfaces.py
    │           ├── __init__.py
    │           └── utils
    │               ├── __init__.py
    │               └── utils.py
    └── README.md
```

Using the collection

This example shows how to use the generated collection in a playbook:

```
----
- hosts: myos101
  gather_facts: False
  tasks:
  - cidrblock.my_collection.myos_interfaces:
    register: result
  - debug:
      var: result
  - cidrblock.my_collection.myos_facts:
  - debug:
      var: ansible_network_resources
```

Using the role

This example shows how to use the generated role in a playbook:

```
- hosts: myos101
  gather_facts: False
  roles:
  - my_role

- hosts: myos101
  gather_facts: False
  tasks:
  - myos_interfaces:
    register: result
  - debug:
      var: result
  - myos_facts:
  - debug:
      var: ansible_network_resources
```

Resource module structure and workflow

The resource module structure includes the following components:

Module

- `library/<ansible_network_os>_<resource>.py`.
- Imports the `module_utils` resource package and calls `execute_module` API:

```
def main():
    result = <resource_package>(module).execute_module()
```

Module argspec

- `module_utils/<ansible_network_os>/argspec/<resource>/`.
- Argspec for the resource.

Facts

- `module_utils/<ansible_network_os>/facts/<resource>/`.
- Populate facts for the resource.
- Entry in `module_utils/<ansible_network_os>/facts/facts.py` for `get_facts` API to keep `<ansible_network_os>_facts` module and facts gathered for the resource module in sync for every subset.
- Entry of Resource subset in FACTS_RESOURCE_SUBSETS list in `module_utils/<ansible_network_os>/facts/facts.py` to make facts collection work.

Module package in module_utils

- `module_utils/<ansible_network_os>/<config>/<resource>/`.
- Implement `execute_module` API that loads the configuration to device and generates the result with `changed`, `commands`, `before` and `after` keys.

- Call get_facts API that returns the <resource> configuration facts or return the difference if the device has onbox diff support.
- Compare facts gathered and given key-values if diff is not supported.
- Generate final configuration.

Utils

- module_utils/<ansible_network_os>/utils.
- Utilities for the <ansible_network_os> platform.

Running ansible-test sanity and tox on resource modules

You should run ansible-test sanity and tox -elinters from the collection root directory before pushing your PR to an Ansible-maintained collection. The CI runs both and will fail if these tests fail. See *Testing Ansible* for details on ansible-test sanity.

To install the necessary packages:

1. Ensure you have a valid Ansible development environment configured. See *Preparing an environment for developing Ansible modules* for details.
2. Run pip install -r requirements.txt from the collection root directory.

   Running tox -elinters:

   - Reads tox.ini from the collection root directory and installs required dependencies (such as black and flake8).
   - Runs these with preconfigured options (such as line-length and ignores.)
   - Runs black in check mode to show which files will be formatted without actually formatting them.

Testing resource modules

The tests rely on a role generated by the resource module builder. After changes to the resource module builder, the role should be regenerated and the tests modified and run as needed. To generate the role after changes:

```
rm -rf rmb_tests/roles/my_role
ansible-playbook -e rm_dest=./rmb_tests/roles/my_role \
                 -e structure=role \
                 -e model=models/myos/interfaces/myos_interfaces.yml \
                 site.yml
```

Resource module integration tests

High-level integration test requirements for new resource modules are as follows:

1. Write a test case for every state.
2. Write additional test cases to test the behavior of the module when an empty config.yaml is given.
3. Add a round trip test case. This involves a merge operation, followed by gather_facts, a merge update with additional configuration, and then reverting back to the base configuration using the previously gathered facts with the state set to overridden.
4. Wherever applicable, assertions should check after and before dicts against a hard coded Source of Truth.

We use Zuul as the CI to run the integration test.

- To view the report, click *Details* on the CI comment in the PR
- To view a failure report, click *ansible/check* and select the failed test.
- To view logs while the test is running, check for your PR number in the Zuul status board.
- To fix static test failure locally, run the **`tox -e black` inside the root folder of collection**.

To view The Ansible run logs and debug test failures:

1. Click the failed job to get the summary, and click *Logs* for the log.
2. Click *console* and scroll down to find the failed test.
3. Click > next to the failed test for complete details.

Integration test structure

Each test case should generally follow this pattern:

- setup —> test —> assert —> test again (for idempotency) —> assert —> tear down (if needed) -> done. This keeps test playbooks from becoming monolithic and difficult to troubleshoot.
- Include a name for each task that is not an assertion. You can add names to assertions as well, but it is easier to identify the broken task within a failed test if you add a name for each task.
- Files containing test cases must end in `.yaml`

Implementation

For platforms that support `connection: local` *and* `connection: network_cli` use the following guidance:

- Name the `targets/` directories after the module name.
- The `main.yaml` file should just reference the transport.

The following example walks through the integration tests for the `vyos.vyos.vyos_l3_interfaces` module in the vyos.vyos collection:

`test/integration/targets/vyos_l3_interfaces/tasks/main.yaml`

```yaml
---
- import_tasks: cli.yaml
  tags:
    - cli
```

`test/integration/targets/vyos_l3_interfaces/tasks/cli.yaml`

```yaml
---
- name: collect all cli test cases
  find:
    paths: "{{ role_path }}/tests/cli"
    patterns: "{{ testcase }}.yaml"
  register: test_cases
  delegate_to: localhost

- name: set test_items
```

(continues on next page)

(continued from previous page)

```
  set_fact: test_items="{{ test_cases.files | map(attribute='path') | list }}"

- name: run test cases (connection=network_cli)
  include_tasks:
     file: "{{ test_case_to_run }}"
  vars:
     ansible_connection: network_cli
  with_items: "{{ test_items }}"
  loop_control:
    loop_var: test_case_to_run

- name: run test case (connection=local)
  include_tasks:
     file: "{{ test_case_to_run }}"
  vars:
     ansible_connection: local
     ansible_become: false
  with_first_found: "{{ test_items }}"
  loop_control:
    loop_var: test_case_to_run
```

test/integration/targets/vyos_l3_interfaces/tests/cli/overridden.yaml

```
---
- debug:
 msg: START vyos_l3_interfaces merged integration tests on connection={{ ansible_
↪connection
   }}

- import_tasks: _remove_config.yaml

- block:

 - import_tasks: _populate.yaml

 - name: Overrides all device configuration with provided configuration
   register: result
   vyos.vyos.vyos_l3_interfaces: &id001
     config:

       - name: eth0
         ipv4:

           - address: dhcp

       - name: eth1
         ipv4:

           - address: 192.0.2.15/24
     state: overridden

 - name: Assert that before dicts were correctly generated
```

(continues on next page)

(continued from previous page)

```
   assert:
     that:
       - "{{ populate | symmetric_difference(result['before']) |length == 0 }}"

 - name: Assert that correct commands were generated
   assert:
     that:
       - "{{ overridden['commands'] | symmetric_difference(result['commands'])\
         \ |length == 0 }}"

 - name: Assert that after dicts were correctly generated
   assert:
     that:
       - "{{ overridden['after'] | symmetric_difference(result['after']) |length\
         \ == 0 }}"

 - name: Overrides all device configuration with provided configurations (IDEMPOTENT)
   register: result
   vyos.vyos.vyos_l3_interfaces: *id001

 - name: Assert that the previous task was idempotent
   assert:
     that:
       - result['changed'] == false

 - name: Assert that before dicts were correctly generated
   assert:
     that:
       - "{{ overridden['after'] | symmetric_difference(result['before']) |length\
         \ == 0 }}"
always:

 - import_tasks: _remove_config.yaml
```

Detecting test resources at runtime

Your tests should detect resources (such as interfaces) at runtime rather than hard-coding them into the test. This allows the test to run on a variety of systems.

For example:

```
- name: Collect interface list
  connection: ansible.netcommon.network_cli
  register: intout
  cisco.nxos.nxos_command:
    commands:
      - show interface brief | json

- set_fact:
    intdataraw: "{{ intout.stdout_lines[0]['TABLE_interface']['ROW_interface'] }}"
```

(continues on next page)

(continued from previous page)

```
- set_fact:
    nxos_int1: '{{ intdataraw[1].interface }}'

- set_fact:
    nxos_int2: '{{ intdataraw[2].interface }}'

- set_fact:
    nxos_int3: '{{ intdataraw[3].interface }}'
```

See the complete test example of this at https://github.com/ansible-collections/cisco.nxos/blob/main/tests/integration/targets/prepare_nxos_tests/tasks/main.yml.

Running network integration tests

Ansible uses Zuul to run an integration test suite on every PR, including new tests introduced by that PR. To find and fix problems in network modules, run the network integration test locally before you submit a PR.

First, create an inventory file that points to your test machines. The inventory group should match the platform name (for example, `eos`, `ios`):

```
cd test/integration
cp inventory.network.template inventory.networking
${EDITOR:-vi} inventory.networking
# Add in machines for the platform(s) you wish to test
```

To run these network integration tests, use `ansible-test network-integration --inventory </path/to/inventory> <tests_to_run>`:

```
ansible-test network-integration  --inventory ~/myinventory -vvv vyos_facts
ansible-test network-integration  --inventory ~/myinventory -vvv vyos_.*
```

To run all network tests for a particular platform:

```
ansible-test network-integration --inventory  /path/to-collection-module/test/
↪integration/inventory.networking vyos_.*
```

This example will run against all `vyos` modules. Note that `vyos_.*` is a regex match, not a bash wildcard - include the . if you modify this example.

To run integration tests for a specific module:

```
ansible-test network-integration --inventory  /path/to-collection-module/test/
↪integration/inventory.networking vyos_l3_interfaces
```

To run a single test case on a specific module:

```
# Only run vyos_l3_interfaces/tests/cli/gathered.yaml
ansible-test network-integration --inventory  /path/to-collection-module/test/
↪integration/inventory.networking vyos_l3_interfaces --testcase gathered
```

To run integration tests for a specific transport:

```
 # Only run nxapi test
ansible-test network-integration --inventory  /path/to-collection-module/test/
↪integration/inventory.networking  --tags="nxapi" nxos_.*

# Skip any cli tests
 ansible-test network-integration --inventory  /path/to-collection-module/test/
↪integration/inventory.networking  --skip-tags="cli" nxos_.*
```

See test/integration/targets/nxos_bgp/tasks/main.yaml for how this is implemented in the tests.

For more options:

```
ansible-test network-integration --help
```

If you need additional help or feedback, reach out in the `#ansible-network` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat).

Unit test requirements

High-level unit test requirements that new resource modules should follow:

1. Write test cases for all the states with all possible combinations of config values.
2. Write test cases to test the error conditions (negative scenarios).
3. Check the value of `changed` and `commands` keys in every test case.

We run all unit test cases on our Zuul test suite, on the latest python version supported by our CI setup.

Use the *same procedure* as the integration tests to view Zuul unit tests reports and logs.

See unit module testing for general unit test details.

Example: Unit testing Ansible network resource modules

This section walks through an example of how to develop unit tests for Ansible resource modules.

See testing_units and testing_units_modules for general documentation on Ansible unit tests for modules. Please read those pages first to understand unit tests and why and when you should use them.

Using mock objects to unit test Ansible network resource modules

Mock objects can be very useful in building unit tests for special or difficult cases, but they can also lead to complex and confusing coding situations. One good use for mocks would be to simulate an API. The `mock` Python package is bundled with Ansible (use `import units.compat.mock`).

You can mock the device connection and output from the device as follows:

```
self.mock_get_config = patch( "ansible_collections.ansible.netcommon.plugins.module_
↪utils.network.common.network.Config.get_config"
)
self.get_config = self.mock_get_config.start()

self.mock_load_config = patch(
"ansible_collections.ansible.netcommon.plugins.module_utils.network.common.network.
```

(continues on next page)

(continued from previous page)

```
↪Config.load_config"
)
self.load_config = self.mock_load_config.start()

self.mock_get_resource_connection_config = patch(
"ansible_collections.ansible.netcommon.plugins.module_utils.network.common.cfg.base.get_
↪resource_connection"
)
self.get_resource_connection_config = (self.mock_get_resource_connection_config.start())

self.mock_get_resource_connection_facts = patch(
"ansible_collections.ansible.netcommon.plugins.module_utils.network.common.facts.facts.
↪get_resource_connection"
)
self.get_resource_connection_facts = (self.mock_get_resource_connection_facts.start())

self.mock_edit_config = patch(
"ansible_collections.arista.eos.plugins.module_utils.network.eos.providers.providers.
↪CliProvider.edit_config"
)
self.edit_config = self.mock_edit_config.start()

self.mock_execute_show_command = patch(
"ansible_collections.arista.eos.plugins.module_utils.network.eos.facts.l2_interfaces.l2_
↪interfaces.L2_interfacesFacts.get_device_data"
)
self.execute_show_command = self.mock_execute_show_command.start()
```

The facts file of the module now includes a new method, `get_device_data`. Call `get_device_data` here to emulate the device output.

Mocking device data

To mock fetching results from devices or provide other complex data structures that come from external libraries, you can use `fixtures` to read in pre-generated data. The text files for this pre-generated data live in `test/units/modules/network/PLATFORM/fixtures/`. See for example the eos_l2_interfaces.cfg file.

Load data using the `load_fixture` method and set this data as the return value of the `get_device_data` method in the facts file:

```
def load_fixtures(self, commands=None, transport='cli'):
    def load_from_file(*args, **kwargs):
        return load_fixture('eos_l2_interfaces_config.cfg')
    self.execute_show_command.side_effect = load_from_file
```

See the unit test file test_eos_l2_interfaces for a practical example.

See also:

testing_units
    Deep dive into developing unit tests for Ansible modules

testing_running_locally
    Running tests locally including gathering and reporting coverage data

Developing modules
Get started developing a module

1.23.2 Developing network plugins

You can extend the existing network modules with custom plugins in your collection.

Network connection plugins

Each network connection plugin has a set of its own plugins which provide a specification of the connection for a particular set of devices. The specific plugin used is selected at runtime based on the value of the `ansible_network_os` variable assigned to the host. This variable should be set to the same value as the name of the plugin to be loaded. Thus, `ansible_network_os=nxos` will try to load a plugin in a file named `nxos.py`, so it is important to name the plugin in a way that will be sensible to users.

Public methods of these plugins may be called from a module or module_utils with the connection proxy object just as other connection methods can. The following is a very simple example of using such a call in a module_utils file so it may be shared with other modules.

```python
from ansible.module_utils.connection import Connection

def get_config(module):
    # module is your AnsibleModule instance.
    connection = Connection(module._socket_path)

    # You can now call any method (that doesn't start with '_') of the connection
    # plugin or its platform-specific plugin
    return connection.get_config()
```

Developing httpapi plugins

httpapi plugins serve as adapters for various HTTP(S) APIs for use with the `httpapi` connection plugin. They should implement a minimal set of convenience methods tailored to the API you are attempting to use.

Specifically, there are a few methods that the `httpapi` connection plugin expects to exist.

Making requests

The `httpapi` connection plugin has a `send()` method, but an httpapi plugin needs a `send_request(self, data, **message_kwargs)` method as a higher-level wrapper to `send()`. This method should prepare requests by adding fixed values like common headers or URL root paths. This method may do more complex work such as turning data into formatted payloads, or determining which path or method to request. It may then also unpack responses to be more easily consumed by the caller.

```
from ansible.module_utils.six.moves.urllib.error import HTTPError

def send_request(self, data, path, method='POST'):
    # Fixed headers for requests
    headers = {'Content-Type': 'application/json'}
    try:
        response, response_content = self.connection.send(path, data, method=method,␣
↪headers=headers)
    except HTTPError as exc:
        return exc.code, exc.read()

    # handle_response (defined separately) will take the format returned by the device
    # and transform it into something more suitable for use by modules.
    # This may be JSON text to Python dictionaries, for example.
    return handle_response(response_content)
```

Authenticating

By default, all requests will authenticate with HTTP Basic authentication. If a request can return some kind of token to stand in place of HTTP Basic, the `update_auth(self, response, response_text)` method should be implemented to inspect responses for such tokens. If the token is meant to be included with the headers of each request, it is sufficient to return a dictionary which will be merged with the computed headers for each request. The default implementation of this method does exactly this for cookies. If the token is used in another way, say in a query string, you should instead save that token to an instance variable, where the `send_request()` method (above) can add it to each request

```
def update_auth(self, response, response_text):
    cookie = response.info().get('Set-Cookie')
    if cookie:
        return {'Cookie': cookie}

    return None
```

If instead an explicit login endpoint needs to be requested to receive an authentication token, the `login(self, username, password)` method can be implemented to call that endpoint. If implemented, this method will be called once before requesting any other resources of the server. By default, it will also be attempted once when a HTTP 401 is returned from a request.

```
def login(self, username, password):
    login_path = '/my/login/path'
    data = {'user': username, 'password': password}

    response = self.send_request(data, path=login_path)
    try:
        # This is still sent as an HTTP header, so we can set our connection's _auth
        # variable manually. If the token is returned to the device in another way,
        # you will have to keep track of it another way and make sure that it is sent
        # with the rest of the request from send_request()
        self.connection._auth = {'X-api-token': response['token']}
    except KeyError:
        raise AnsibleAuthenticationFailure(message="Failed to acquire login token.")
```

Similarly, `logout(self)` can be implemented to call an endpoint to invalidate and/or release the current token, if such an endpoint exists. This will be automatically called when the connection is closed (and, by extension, when reset).

```
def logout(self):
    logout_path = '/my/logout/path'
    self.send_request(None, path=logout_path)

    # Clean up tokens
    self.connection._auth = None
```

Error handling

The `handle_httperror(self, exception)` method can deal with status codes returned by the server. The return value indicates how the plugin will continue with the request:

- A value of `true` means that the request can be retried. This my be used to indicate a transient error, or one that has been resolved. For example, the default implementation will try to call `login()` when presented with a 401, and return `true` if successful.
- A value of `false` means that the plugin is unable to recover from this response. The status code will be raised as an exception to the calling module.
- Any other value will be taken as a nonfatal response from the request. This may be useful if the server returns error messages in the body of the response. Returning the original exception is usually sufficient in this case, as HTTPError objects have the same interface as a successful response.

For example httpapi plugins, see the source code for the httpapi plugins included with Ansible Core.

Developing NETCONF plugins

The netconf connection plugin provides a connection to remote devices over the `SSH NETCONF` subsystem. Network devices typically use this connection plugin to send and receive `RPC` calls over `NETCONF`.

The `netconf` connection plugin uses the `ncclient` Python library under the hood to initiate a NETCONF session with a NETCONF-enabled remote network device. `ncclient` also executes NETCONF RPC requests and receives responses. You must install the `ncclient` on the local Ansible controller.

To use the `netconf` connection plugin for network devices that support standard NETCONF (**RFC 6241**) operations such as `get`, `get-config`, `edit-config`, set `ansible_network_os=default`. You can use netconf_get, netconf_config and netconf_rpc modules to talk to a NETCONF enabled remote host.

As a contributor and user, you should be able to use all the methods under the `NetconfBase` class if your device supports standard NETCONF. You can contribute a new plugin if the device you are working with has a vendor specific NETCONF RPC. To support a vendor specific NETCONF RPC, add the implementation in the network OS specific NETCONF plugin.

For Junos for example:

- See the vendor-specific Junos RPC methods implemented in `plugins/netconf/junos.py`.
- Set the value of `ansible_network_os` to the name of the netconf plugin file, that is `junos` in this case.

Developing network_cli plugins

The network_cli connection type uses `paramiko_ssh` under the hood which creates a pseudo terminal to send commands and receive responses. `network_cli` loads two platform specific plugins based on the value of `ansible_network_os`:

- Terminal plugin (for example `plugins/terminal/ios.py`) - Controls the parameters related to terminal, such as setting terminal length and width, page disabling and privilege escalation. Also defines regex to identify the command prompt and error prompts.
- *Cliconf plugins* (for example, ios cliconf) - Provides an abstraction layer for low level send and receive operations. For example, the `edit_config()` method ensures that the prompt is in `config` mode before executing configuration commands.

To contribute a new network operating system to work with the `network_cli` connection, implement the `cliconf` and `terminal` plugins for that network OS.

The plugins can reside in:

- Adjacent to playbook in folders

```
cliconf_plugins/
terminal_plugins/
```

- Roles

```
myrole/cliconf_plugins/
myrole/terminal_plugins/
```

- Collections

```
myorg/mycollection/plugins/terminal/
myorg/mycollection/plugins/cliconf/
```

The user can also set the *DEFAULT_CLICONF_PLUGIN_PATH* to configure the `cliconf` plugin path.

After adding the `cliconf` and `terminal` plugins in the expected locations, users can:

- Use the cli_command to run an arbitrary command on the network device.
- Use the cli_config to implement configuration changes on the remote hosts without platform-specific modules.

Developing cli_parser plugins in a collection

You can use `cli_parse` as an entry point for a cli_parser plugin in your own collection.

The following sample shows the start of a custom cli_parser plugin:

```
from ansible_collections.ansible.netcommon.plugins.module_utils.cli_parser.cli_
→parserbase import (
    CliParserBase,
)

class CliParser(CliParserBase):
    """ Sample cli_parser plugin
    """

    # Use the follow extension when loading a template
    DEFAULT_TEMPLATE_EXTENSION = "txt"
    # Provide the contents of the template to the parse function
    PROVIDE_TEMPLATE_CONTENTS = True

    def myparser(text, template_contents):
      # parse the text using the template contents
      return {...}

    def parse(self, *_args, **kwargs):
        """ Standard entry point for a cli_parse parse execution

        :return: Errors or parsed text as structured data
        :rtype: dict

        :example:

        The parse function of a parser should return a dict:
        {"errors": [a list of errors]}
        or
        {"parsed": obj}
        """
        template_contents = kwargs["template_contents"]
        text = self._task_args.get("text")
        try:
            parsed = myparser(text, template_contents)
        except Exception as exc:
            msg = "Custom parser returned an error while parsing. Error: {err}"
            return {"errors": [msg.format(err=to_native(exc))]}
        return {"parsed": parsed}
```

The following task uses this custom cli_parser plugin:

```
- name: Use a custom cli_parser
  ansible.netcommon.cli_parse:
    command: ls -l
    parser:
      name: my_organiztion.my_collection.custom_parser
```

To develop a custom plugin: - Each cli_parser plugin requires a `CliParser` class. - Each cli_parser plugin requires a `parse` function. - Always return a dictionary with `errors` or `parsed`. - Place the custom cli_parser in plugins/cli_parsers directory of the collection. - See the current cli_parsers for examples to follow.

See also:

- *Parsing semi-structured text with Ansible*

1.23.3 Documenting new network platforms

When you create network modules for a new platform, or modify the connections provided by an existing network platform (such as `network_cli` and `httpapi`), you also need to update the *Settings by Platform* table and add or modify the Platform Options file for your platform.

You should already have documented each module as described in *Module format and documentation*.

Modifying the platform options table

The *Settings by Platform* table is a convenient summary of the connections options provided by each network platform that has modules in Ansible. Add a row for your platform to this table, in alphabetical order. For example:

```
+-------------------+-------------------------+-------------+---------+---------+--------
↪--+
| My OS             | ``myos``                | ✓           | ✓       |         | ✓      ␣
↪   |
```

Ensure that the table stays formatted correctly. That is:

- Each row is inserted in alphabetical order.
- The cell division `|` markers line up with the `+` markers.
- The check marks appear only for the connection types provided by the network modules.

Adding a platform-specific options section

The platform- specific sections are individual `.rst` files that provide more detailed information for the users of your network platform modules. Name your new file `platform_<name>.rst` (for example, `platform_myos.rst`). The platform name should match the module prefix. See platform_eos.rst and *EOS Platform Options* for an example of the details you should provide in your platform-specific options section.

Your platform-specific section should include the following:

- **Connections available table** - a deeper dive into each connection type, including details on credentials, indirect access, connections settings, and enable mode.
- **How to use each connection type** - with working examples of each connection type.

If your network platform supports SSH connections, also include the following at the bottom of your `.rst` file:

```
.. include:: shared_snippets/SSH_warning.txt
```

Adding your new file to the table of contents

As a final step, add your new file in alphabetical order in the `platform_index.rst` file. You should then build the documentation to verify your additions. See *Contributing to the Ansible Documentation* for more details.

1.24 Galaxy User Guide

Ansible Galaxy refers to the Galaxy website, a free site for finding, downloading, and sharing community developed roles.

Use Galaxy to jump-start your automation project with great content from the Ansible community. Galaxy provides pre-packaged units of work such as *roles*, and new in Galaxy 3.2, *collections* You can find roles for provisioning infrastructure, deploying applications, and all of the tasks you do everyday. The collection format provides a comprehensive package of automation that may include multiple playbooks, roles, modules, and plugins.

1.24.1 Finding collections on Galaxy

To find collections on Galaxy:

1. Click the *Search* icon in the left-hand navigation.
2. Set the filter to *collection*.
3. Set other filters and press *enter*.

Galaxy presents a list of collections that match your search criteria.

1.24.2 Installing collections

Installing a collection from Galaxy

By default, `ansible-galaxy collection install` uses https://galaxy.ansible.com as the Galaxy server (as listed in the `ansible.cfg` file under *GALAXY_SERVER*). You do not need any further configuration.

See *Configuring the ansible-galaxy client* if you are using any other Galaxy server, such as Red Hat Automation Hub.

To install a collection hosted in Galaxy:

```
ansible-galaxy collection install my_namespace.my_collection
```

To upgrade a collection to the latest available version from the Galaxy server you can use the `--upgrade` option:

```
ansible-galaxy collection install my_namespace.my_collection --upgrade
```

You can also directly use the tarball from your build:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./
↪collections
```

You can build and install a collection from a local source directory. The `ansible-galaxy` utility builds the collection using the `MANIFEST.json` or `galaxy.yml` metadata in the directory.

```
ansible-galaxy collection install /path/to/collection -p ./collections
```

You can also install multiple collections in a namespace directory.

```
ns/
├── collection1/
│   ├── MANIFEST.json
│   └── plugins/
└── collection2/
    ├── galaxy.yml
    └── plugins/
```

```
ansible-galaxy collection install /path/to/ns -p ./collections
```

Note: The install command automatically appends the path `ansible_collections` to the one specified with the `-p` option unless the parent directory is already in a folder called `ansible_collections`.

When using the `-p` option to specify the install path, use one of the values configured in *COLLECTIONS_PATHS*, as this is where Ansible itself will expect to find collections. If you don't specify a path, `ansible-galaxy collection install` installs the collection to the first path defined in *COLLECTIONS_PATHS*, which by default is `~/.ansible/collections`

You can also keep a collection adjacent to the current playbook, under a `collections/ansible_collections/` directory structure.

```
./
├── play.yml
├── collections/
│   └── ansible_collections/
│               └── my_namespace/
│                   └── my_collection/<collection structure lives here>
```

See *Collection structure* for details on the collection directory structure.

Downloading a collection from Automation Hub

You can download collections from Automation Hub at the command line. Automation Hub content is available to subscribers only, so you must download an API token and configure your local environment to provide it before you can you download collections. To download a collection from Automation Hub with the `ansible-galaxy` command:

1. Get your Automation Hub API token. Go to https://cloud.redhat.com/ansible/automation-hub/token/ and click *Load token* from the version dropdown to copy your API token.
2. Configure Red Hat Automation Hub server in the `server_list` option under the `[galaxy]` section in your `ansible.cfg` file.

   ```
   [galaxy]
   server_list = automation_hub

   [galaxy_server.automation_hub]
   url=https://console.redhat.com/api/automation-hub/
   auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
   ↪connect/token
   token=my_ah_token
   ```

3. Download the collection hosted in Automation Hub.

   ```
   ansible-galaxy collection install my_namespace.my_collection
   ```

See also:

Getting started with Automation Hub
An introduction to Automation Hub

Installing an older version of a collection

You can only have one version of a collection installed at a time. By default `ansible-galaxy` installs the latest available version. If you want to install a specific version, you can add a version range identifier. For example, to install the 1.0.0-beta.1 version of the collection:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.0-beta.1
```

You can specify multiple range identifiers separated by `,`. Use single quotes so the shell passes the entire command, including >, `!`, and other operators, along. For example, to install the most recent version that is greater than or equal to 1.0.0 and less than 2.0.0:

```
ansible-galaxy collection install 'my_namespace.my_collection:>=1.0.0,<2.0.0'
```

Ansible will always install the most recent version that meets the range identifiers you specify. You can use the following range identifiers:

- `*`: The most recent version. This is the default.
- `!=`: Not equal to the version specified.
- ==: Exactly the version specified.
- >=: Greater than or equal to the version specified.
- >: Greater than the version specified.
- <=: Less than or equal to the version specified.
- <: Less than the version specified.

Note: By default `ansible-galaxy` ignores pre-release versions. To install a pre-release version, you must use the == range identifier to require it explicitly.

Install multiple collections with a requirements file

You can set up a `requirements.yml` file to install multiple collections in one command. This file is a YAML file in the format:

```
---
collections:
# With just the collection name
- my_namespace.my_collection

# With the collection name, version, and source options
- name: my_namespace.my_other_collection
  version: ">=1.2.0" # Version range identifiers (default: ``*``)
  source: ... # The Galaxy URL to pull the collection from (default: ``--api-server``␣
↪from cmdline)
```

You can specify the following keys for each collection entry:

- `name`
- `version`
- `signatures`

- source
- type

The version key uses the same range identifier format documented in *Installing an older version of a collection*.

The signatures key accepts a list of signature sources that are used to supplement those found on the Galaxy server during collection installation and `ansible-galaxy collection verify`. Signature sources should be URIs that contain the detached signature. The `--keyring` CLI option must be provided if signatures are specified.

Signatures are only used to verify collections on Galaxy servers. User-provided signatures are not used to verify collections installed from git repositories, source directories, or URLs/paths to tar.gz files.

```
collections:
  - name: namespace.name
    version: 1.0.0
    type: galaxy
    signatures:
      - https://examplehost.com/detached_signature.asc
      - file:///path/to/local/detached_signature.asc
```

The `type` key can be set to `file`, `galaxy`, `git`, `url`, `dir`, or `subdirs`. If `type` is omitted, the `name` key is used to implicitly determine the source of the collection.

When you install a collection with `type: git`, the `version` key can refer to a branch or to a git commit-ish object (commit or tag). For example:

```
collections:
  - name: https://github.com/organization/repo_name.git
    type: git
    version: devel
```

You can also add roles to a `requirements.yml` file, under the `roles` key. The values follow the same format as a requirements file used in older Ansible releases.

```
---
roles:
  # Install a role from Ansible Galaxy.
  - name: geerlingguy.java
    version: "1.9.6" # note that ranges are not supported for roles


collections:
  # Install a collection from Ansible Galaxy.
  - name: geerlingguy.php_roles
    version: ">=0.9.3"
    source: https://galaxy.ansible.com
```

To install both roles and collections at the same time with one command, run the following:

```
$ ansible-galaxy install -r requirements.yml
```

Running `ansible-galaxy collection install -r` or `ansible-galaxy role install -r` will only install collections, or roles respectively.

Note: Installing both roles and collections from the same requirements file will not work when specifying a custom collection or role install path. In this scenario the collections will be skipped and the command will process each like `ansible-galaxy role install` would.

Downloading a collection for offline use

To download the collection tarball from Galaxy for offline use:

1. Navigate to the collection page.
2. Click on *Download tarball*.

You may also need to manually download any dependent collections.

Installing a collection from source files

Ansible can also install from a source directory in several ways:

```
collections:
  # directory containing the collection
  - source: ./my_namespace/my_collection/
    type: dir

  # directory containing a namespace, with collections as subdirectories
  - source: ./my_namespace/
    type: subdirs
```

Ansible can also install a collection collected with `ansible-galaxy collection build` or downloaded from Galaxy for offline use by specifying the output file directly:

```
collections:
  - name: /tmp/my_namespace-my_collection-1.0.0.tar.gz
    type: file
```

Note: Relative paths are calculated from the current working directory (where you are invoking `ansible-galaxy install -r` from). They are not taken relative to the `requirements.yml` file.

Installing a collection from a git repository

You can install a collection from a git repository instead of from Galaxy or Automation Hub. As a developer, installing from a git repository lets you review your collection before you create the tarball and publish the collection. As a user, installing from a git repository lets you use collections or versions that are not in Galaxy or Automation Hub yet.

The repository must contain a `galaxy.yml` or `MANIFEST.json` file. This file provides metadata such as the version number and namespace of the collection.

Installing a collection from a git repository at the command line

To install a collection from a git repository at the command line, use the URI of the repository instead of a collection name or path to a `tar.gz` file. Use the prefix `git+`, unless you're using SSH authentication with the user `git` (for example, `git@github.com:ansible-collections/ansible.windows.git`). You can specify a branch, commit, or tag using the comma-separated git commit-ish syntax.

For example:

```
# Install a collection in a repository using the latest commit on the branch 'devel'
ansible-galaxy collection install git+https://github.com/organization/repo_name.git,devel

# Install a collection from a private github repository
ansible-galaxy collection install git@github.com:organization/repo_name.git

# Install a collection from a local git repository
ansible-galaxy collection install git+file:///home/user/path/to/repo_name.git
```

Warning: Embedding credentials into a git URI is not secure. Use safe authentication options to prevent your credentials from being exposed in logs or elsewhere.

- Use SSH authentication
- Use netrc authentication
- Use http.extraHeader in your git configuration
- Use url.<base>.pushInsteadOf in your git configuration

Specifying the collection location within the git repository

When you install a collection from a git repository, Ansible uses the collection `galaxy.yml` or `MANIFEST.json` metadata file to build the collection. By default, Ansible searches two paths for collection `galaxy.yml` or `MANIFEST.json` metadata files:

- The top level of the repository.
- Each directory in the repository path (one level deep).

If a `galaxy.yml` or `MANIFEST.json` file exists in the top level of the repository, Ansible uses the collection metadata in that file to install an individual collection.

```
├── galaxy.yml
├── plugins/
│   ├── lookup/
│   ├── modules/
│   └── module_utils/
└─── README.md
```

If a `galaxy.yml` or `MANIFEST.json` file exists in one or more directories in the repository path (one level deep), Ansible installs each directory with a metadata file as a collection. For example, Ansible installs both collection1 and collection2 from this repository structure by default:

```
├── collection1
│   ├── docs/
│   ├── galaxy.yml
│   └── plugins/
│       ├── inventory/
│       └── modules/
└── collection2
    ├── docs/
    ├── galaxy.yml
    ├── plugins/
    |   ├── filter/
    |   └── modules/
    └── roles/
```

If you have a different repository structure or only want to install a subset of collections, you can add a fragment to the end of your URI (before the optional comma-separated version) to indicate the location of the metadata file or files. The path should be a directory, not the metadata file itself. For example, to install only collection2 from the example repository with two collections:

```
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪collection2/
```

In some repositories, the main directory corresponds to the namespace:

```
namespace/
├── collectionA/
|   ├── docs/
|   ├── galaxy.yml
|   ├── plugins/
|   │   ├── README.md
|   │   └── modules/
|   ├── README.md
|   └── roles/
└── collectionB/
    ├── docs/
    ├── galaxy.yml
    ├── plugins/
    │   ├── connection/
    │   └── modules/
    ├── README.md
    └── roles/
```

You can install all collections in this repository, or install one collection from a specific commit:

```
# Install all collections in the namespace
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪namespace/

# Install an individual collection using a specific commit
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/
↪namespace/collectionA/,7b60ddc245bc416b72d8ea6ed7b799885110f5e5
```

Listing installed collections

To list installed collections, run `ansible-galaxy collection list`. See *Listing collections* for more details.

Configuring the `ansible-galaxy` client

By default, `ansible-galaxy` uses https://galaxy.ansible.com as the Galaxy server (as listed in the `ansible.cfg` file under *GALAXY_SERVER*).

You can use either option below to configure `ansible-galaxy collection` to use other servers (such as a custom Galaxy server):

- Set the server list in the *GALAXY_SERVER_LIST* configuration option in *The configuration file*.
- Use the `--server` command line argument to limit to an individual server.

To configure a Galaxy server list in `ansible.cfg`:

1. Add the `server_list` option under the `[galaxy]` section to one or more server names.
2. Create a new section for each server name.
3. Set the `url` option for each server name.
4. Optionally, set the API token for each server name. Go to https://galaxy.ansible.com/me/preferences and click *Show API key*.

Note: The `url` option for each server name must end with a forward slash `/`. If you do not set the API token in your Galaxy server list, use the `--api-key` argument to pass in the token to the `ansible-galaxy collection publish` command.

The following example shows how to configure multiple servers:

```
[galaxy]
server_list = my_org_hub, release_galaxy, test_galaxy, my_galaxy_ng

[galaxy_server.my_org_hub]
url=https://automation.my_org/
username=my_user
password=my_pass

[galaxy_server.release_galaxy]
url=https://galaxy.ansible.com/
token=my_token

[galaxy_server.test_galaxy]
url=https://galaxy-dev.ansible.com/
token=my_test_token

[galaxy_server.my_galaxy_ng]
url=http://my_galaxy_ng:8000/api/automation-hub/
auth_url=http://my_keycloak:8080/auth/realms/myco/protocol/openid-connect/token
client_id=galaxy-ng
token=my_keycloak_access_token
```

Note: You can use the `--server` command line argument to select an explicit Galaxy server in the `server_list` and the value of this argument should match the name of the server. To use a server not in the server list, set the value to the URL to access that server (all servers in the server list will be ignored). Also you cannot use the `--api-key` argument for any of the predefined servers. You can only use the `api_key` argument if you did not define a server list or if you specify a URL in the `--server` argument.

Galaxy server list configuration options

The *GALAXY_SERVER_LIST* option is a list of server identifiers in a prioritized order. When searching for a collection, the install process will search in that order, for example, `automation_hub` first, then `my_org_hub`, `release_galaxy`, and finally `test_galaxy` until the collection is found. The actual Galaxy instance is then defined under the section `[galaxy_server.{{ id }}]` where `{{ id }}` is the server identifier defined in the list. This section can then define the following keys:

- `url`: The URL of the Galaxy instance to connect to. Required.
- `token`: An API token key to use for authentication against the Galaxy instance. Mutually exclusive with `username`.
- `username`: The username to use for basic authentication against the Galaxy instance. Mutually exclusive with `token`.
- `password`: The password to use, in conjunction with `username`, for basic authentication.
- `auth_url`: The URL of a Keycloak server 'token_endpoint' if using SSO authentication (for example, galaxyNG). Mutually exclusive with `username`. Requires `token`.
- `validate_certs`: Whether or not to verify TLS certificates for the Galaxy server. This defaults to True unless the `--ignore-certs` option is provided or `GALAXY_IGNORE_CERTS` is configured to True.
- `client_id`: The Keycloak token's client_id to use for authentication. Requires `auth_url` and `token`. The default `client_id` is cloud-services to work with Red Hat SSO.

As well as defining these server options in the `ansible.cfg` file, you can also define them as environment variables. The environment variable is in the form `ANSIBLE_GALAXY_SERVER_{{ id }}_{{ key }}` where `{{ id }}` is the upper case form of the server identifier and `{{ key }}` is the key to define. For example I can define `token` for `release_galaxy` by setting `ANSIBLE_GALAXY_SERVER_RELEASE_GALAXY_TOKEN=secret_token`.

For operations that use only one Galaxy server (for example, the `publish`, `info`, or `install` commands). the `ansible-galaxy collection` command uses the first entry in the `server_list`, unless you pass in an explicit server with the `--server` argument.

Note: `ansible-galaxy` can seek out dependencies on other configured Galaxy instances to support the use case where a collection can depend on a collection from another Galaxy instance.

1.24.3 Finding roles on Galaxy

Search the Galaxy database by tags, platforms, author and multiple keywords. For example:

```
$ ansible-galaxy search elasticsearch --author geerlingguy
```

The search command will return a list of the first 1000 results matching your search:

```
Found 2 roles matching your search:

Name                             Description
----                             -----------
geerlingguy.elasticsearch        Elasticsearch for Linux.
geerlingguy.elasticsearch-curator Elasticsearch curator for Linux.
```

Get more information about a role

Use the `info` command to view more detail about a specific role:

```
$ ansible-galaxy info username.role_name
```

This returns everything found in Galaxy for the role:

```
Role: username.role_name
    description: Installs and configures a thing, a distributed, highly available NoSQL␣
↪thing.
    active: True
    commit: c01947b7bc89ebc0b8a2e298b87ab416aed9dd57
    commit_message: Adding travis
    commit_url: https://github.com/username/repo_name/commit/
↪c01947b7bc89ebc0b8a2e298b87ab
    company: My Company, Inc.
    created: 2015-12-08T14:17:52.773Z
    download_count: 1
    forks_count: 0
    github_branch:
    github_repo: repo_name
    github_user: username
    id: 6381
    is_valid: True
    issue_tracker_url:
    license: Apache
    min_ansible_version: 1.4
    modified: 2015-12-08T18:43:49.085Z
    namespace: username
    open_issues_count: 0
    path: /Users/username/projects/roles
    scm: None
    src: username.repo_name
    stargazers_count: 0
    travis_status_url: https://travis-ci.org/username/repo_name.svg?branch=main
    version:
    watchers_count: 1
```

1.24.4 Installing roles from Galaxy

The `ansible-galaxy` command comes bundled with Ansible, and you can use it to install roles from Galaxy or directly from a git based SCM. You can also use it to create a new role, remove roles, or perform tasks on the Galaxy website.

The command line tool by default communicates with the Galaxy website API using the server address *https://galaxy.ansible.com*. If you run your own internal Galaxy server and want to use it instead of the default one, pass the `--server` option following the address of this galaxy server. You can set permanently this option by setting the Galaxy server value in your `ansible.cfg` file to use it . For information on setting the value in *ansible.cfg* see *GALAXY_SERVER*.

Installing roles

Use the `ansible-galaxy` command to download roles from the Galaxy website

```
$ ansible-galaxy install namespace.role_name
```

Setting where to install roles

By default, Ansible downloads roles to the first writable directory in the default list of paths `~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`. This installs roles in the home directory of the user running `ansible-galaxy`.

You can override this with one of the following options:

- Set the environment variable *`ANSIBLE_ROLES_PATH`* in your session.
- Use the `--roles-path` option for the `ansible-galaxy` command.
- Define `roles_path` in an `ansible.cfg` file.

The following provides an example of using `--roles-path` to install the role into the current working directory:

```
$ ansible-galaxy install --roles-path . geerlingguy.apache
```

See also:

Configuring Ansible
All about configuration files

Installing a specific version of a role

When the Galaxy server imports a role, it imports any git tags matching the Semantic Version format as versions. In turn, you can download a specific version of a role by specifying one of the imported tags.

To see the available versions for a role:

1. Locate the role on the Galaxy search page.
2. Click on the name to view more details, including the available versions.

You can also navigate directly to the role using the /<namespace>/<role name>. For example, to view the role geerlingguy.apache, go to https://galaxy.ansible.com/geerlingguy/apache.

To install a specific version of a role from Galaxy, append a comma and the value of a GitHub release tag. For example:

```
$ ansible-galaxy install geerlingguy.apache,1.0.0
```

It is also possible to point directly to the git repository and specify a branch name or commit hash as the version. For example, the following will install a specific commit:

```
$ ansible-galaxy install git+https://github.com/geerlingguy/ansible-role-apache.git,
↪0b7cd353c0250e87a26e0499e59e7fd265cc2f25
```

Installing multiple roles from a file

You can install multiple roles by including the roles in a `requirements.yml` file. The format of the file is YAML, and the file extension must be either *.yml* or *.yaml*.

Use the following command to install roles included in `requirements.yml`:

```
$ ansible-galaxy install -r requirements.yml
```

Again, the extension is important. If the *.yml* extension is left off, the `ansible-galaxy` CLI assumes the file is in an older, now deprecated, "basic" format.

Each role in the file will have one or more of the following attributes:

src
: The source of the role. Use the format *namespace.role_name*, if downloading from Galaxy; otherwise, provide a URL pointing to a repository within a git based SCM. See the examples below. This is a required attribute.

scm
: Specify the SCM. As of this writing only *git* or *hg* are allowed. See the examples below. Defaults to *git*.

version:
: The version of the role to download. Provide a release tag value, commit hash, or branch name. Defaults to the branch set as a default in the repository, otherwise defaults to the *master*.

name:
: Download the role to a specific name. Defaults to the Galaxy name when downloading from Galaxy, otherwise it defaults to the name of the repository.

Use the following example as a guide for specifying roles in *requirements.yml*:

```
# from galaxy
- name: yatesr.timezone

# from locally cloned git repository (git+file:// requires full paths)
- src: git+file:///home/bennojoy/nginx

# from GitHub
- src: https://github.com/bennojoy/nginx

# from GitHub, overriding the name and specifying a specific tag
- name: nginx_role
  src: https://github.com/bennojoy/nginx
  version: main

# from GitHub, specifying a specific commit hash
- src: https://github.com/bennojoy/nginx
  version: "ee8aa41"
```

(continues on next page)

(continued from previous page)

```
# from a webserver, where the role is packaged in a tar.gz
- name: http-role-gz
  src: https://some.webserver.example.com/files/main.tar.gz

# from a webserver, where the role is packaged in a tar.bz2
- name: http-role-bz2
  src: https://some.webserver.example.com/files/main.tar.bz2

# from a webserver, where the role is packaged in a tar.xz (Python 3.x only)
- name: http-role-xz
  src: https://some.webserver.example.com/files/main.tar.xz

# from Bitbucket
- src: git+https://bitbucket.org/willthames/git-ansible-galaxy
  version: v1.4

# from Bitbucket, alternative syntax and caveats
- src: https://bitbucket.org/willthames/hg-ansible-galaxy
  scm: hg

# from GitLab or other git-based scm, using git+ssh
- src: git@gitlab.company.com:mygroup/ansible-core.git
  scm: git
  version: "0.1"  # quoted, so YAML doesn't parse this as a floating-point value
```

Warning: Embedding credentials into a SCM URL is not secure. Make sure to use safe auth options for security reasons. For example, use SSH, netrc or http.extraHeader/url.<base>.pushInsteadOf in Git config to prevent your creds from being exposed in logs.

Installing roles and collections from the same requirements.yml file

You can install roles and collections from the same requirements files

```
---
roles:
  # Install a role from Ansible Galaxy.
  - name: geerlingguy.java
    version: "1.9.6" # note that ranges are not supported for roles

collections:
  # Install a collection from Ansible Galaxy.
  - name: geerlingguy.php_roles
    version: ">=0.9.3"
    source: https://galaxy.ansible.com
```

Installing multiple roles from multiple files

For large projects, the `include` directive in a `requirements.yml` file provides the ability to split a large file into multiple smaller files.

For example, a project may have a `requirements.yml` file, and a `webserver.yml` file.

Below are the contents of the `webserver.yml` file:

```
# from github
- src: https://github.com/bennojoy/nginx

# from Bitbucket
- src: git+https://bitbucket.org/willthames/git-ansible-galaxy
  version: v1.4
```

The following shows the contents of the `requirements.yml` file that now includes the `webserver.yml` file:

```
# from galaxy
- name: yatesr.timezone
- include: <path_to_requirements>/webserver.yml
```

To install all the roles from both files, pass the root file, in this case `requirements.yml` on the command line, as follows:

```
$ ansible-galaxy install -r requirements.yml
```

Dependencies

Roles can also be dependent on other roles, and when you install a role that has dependencies, those dependencies will automatically be installed to the `roles_path`.

There are two ways to define the dependencies of a role:

- using `meta/requirements.yml`
- using `meta/main.yml`

Using `meta/requirements.yml`

New in version 2.10.

You can create the file `meta/requirements.yml` and define dependencies in the same format used for `requirements.yml` described in the *Installing multiple roles from a file* section.

From there, you can import or include the specified roles in your tasks.

Using meta/main.yml

Alternatively, you can specify role dependencies in the meta/main.yml file by providing a list of roles under the dependencies section. If the source of a role is Galaxy, you can simply specify the role in the format namespace.role_name. You can also use the more complex format in requirements.yml, allowing you to provide src, scm, version, and name.

Dependencies installed that way, depending on other factors described below, will also be executed **before** this role is executed during play execution. To better understand how dependencies are handled during play execution, see *Roles*.

The following shows an example meta/main.yml file with dependent roles:

```yaml
---
dependencies:
  - geerlingguy.java

galaxy_info:
  author: geerlingguy
  description: Elasticsearch for Linux.
  company: "Midwestern Mac, LLC"
  license: "license (BSD, MIT)"
  min_ansible_version: 2.4
  platforms:
  - name: EL
    versions:
    - all
  - name: Debian
    versions:
    - all
  - name: Ubuntu
    versions:
    - all
  galaxy_tags:
    - web
    - system
    - monitoring
    - logging
    - lucene
    - elk
    - elasticsearch
```

Tags are inherited *down* the dependency chain. In order for tags to be applied to a role and all its dependencies, the tag should be applied to the role, not to all the tasks within a role.

Roles listed as dependencies are subject to conditionals and tag filtering, and may not execute fully depending on what tags and conditionals are applied.

If the source of a role is Galaxy, specify the role in the format *namespace.role_name*:

```yaml
dependencies:
  - geerlingguy.apache
  - geerlingguy.ansible
```

Alternately, you can specify the role dependencies in the complex form used in requirements.yml as follows:

```
dependencies:
  - name: geerlingguy.ansible
  - name: composer
    src: git+https://github.com/geerlingguy/ansible-role-composer.git
    version: 775396299f2da1f519f0d8885022ca2d6ee80ee8
```

Note: Galaxy expects all role dependencies to exist in Galaxy, and therefore dependencies to be specified in the `namespace.role_name` format. If you import a role with a dependency where the `src` value is a URL, the import process will fail.

List installed roles

Use `list` to show the name and version of each role installed in the *roles_path*.

```
$ ansible-galaxy list
  - ansible-network.network-engine, v2.7.2
  - ansible-network.config_manager, v2.6.2
  - ansible-network.cisco_nxos, v2.7.1
  - ansible-network.vyos, v2.7.3
  - ansible-network.cisco_ios, v2.7.0
```

Remove an installed role

Use `remove` to delete a role from *roles_path*:

```
$ ansible-galaxy remove namespace.role_name
```

See also:

Using Ansible collections
    Shareable collections of modules, playbooks and roles

Roles
    Reusable tasks, handlers, and other files in a known directory structure

Working with command line tools
    Perform other related operations

1.25 Galaxy Developer Guide

You can host collections and roles on Galaxy to share with the Ansible community. Galaxy content is formatted in pre-packaged units of work such as *roles*, and new in Galaxy 3.2, *collections*. You can create roles for provisioning infrastructure, deploying applications, and all of the tasks you do everyday. Taking this a step further, you can create collections which provide a comprehensive package of automation that may include multiple playbooks, roles, modules, and plugins.

1.25.1 Creating collections for Galaxy

Collections are a distribution format for Ansible content. You can use collections to package and distribute playbooks, roles, modules, and plugins. You can publish and use collections through Ansible Galaxy.

See *Developing collections* for details on how to create collections.

1.25.2 Creating roles for Galaxy

Use the `init` command to initialize the base structure of a new role, saving time on creating the various directories and main.yml files a role requires

```
$ ansible-galaxy init role_name
```

The above will create the following directory structure in the current working directory:

```
role_name/
    README.md
    .travis.yml
    defaults/
        main.yml
    files/
    handlers/
        main.yml
    meta/
        main.yml
    templates/
    tests/
        inventory
        test.yml
    vars/
        main.yml
```

If you want to create a repository for the role, the repository root should be *role_name*.

Force

If a directory matching the name of the role already exists in the current working directory, the init command will result in an error. To ignore the error use the `--force` option. Force will create the above subdirectories and files, replacing anything that matches.

Container enabled

If you are creating a Container Enabled role, pass `--type container` to `ansible-galaxy init`. This will create the same directory structure as above, but populate it with default files appropriate for a Container Enabled role. For instance, the README.md has a slightly different structure, the *.travis.yml* file tests the role using Ansible Container, and the meta directory includes a *container.yml* file.

Using a custom role skeleton

A custom role skeleton directory can be supplied as follows:

```
$ ansible-galaxy init --role-skeleton=/path/to/skeleton role_name
```

When a skeleton is provided, init will:

- copy all files and directories from the skeleton to the new role
- any .j2 files found outside of a templates folder will be rendered as templates. The only useful variable at the moment is role_name
- The .git folder and any .git_keep files will not be copied

Alternatively, the role_skeleton and ignoring of files can be configured via ansible.cfg

```
[galaxy]
role_skeleton = /path/to/skeleton
role_skeleton_ignore = ^.git$,^.*/.git_keep$
```

Authenticate with Galaxy

Using the `import`, `delete` and `setup` commands to manage your roles on the Galaxy website requires authentication in the form of an API key, you must create an account on the Galaxy website.

1. Log in to the Galaxy website and open the Preferences view.
2. Select **Show API key** and then copy it.
3. Save your token in the path set in the *GALAXY_TOKEN_PATH*.

Import a role

The `import` command requires that you first authenticate using the `login` command. Once authenticated you can import any GitHub repository that you own or have been granted access.

Use the following to import to role:

```
$ ansible-galaxy import github_user github_repo
```

By default the command will wait for Galaxy to complete the import process, displaying the results as the import progresses:

```
Successfully submitted import request 41
Starting import 41: role_name=myrole repo=githubuser/ansible-role-repo ref=
Retrieving GitHub repo githubuser/ansible-role-repo
Accessing branch: devel
Parsing and validating meta/main.yml
Parsing galaxy_tags
Parsing platforms
Adding dependencies
Parsing and validating README.md
Adding repo tags as role versions
Import completed
Status SUCCESS : warnings=0 errors=0
```

Branch

Use the `--branch` option to import a specific branch. If not specified, the default branch for the repo will be used.

Role name

By default the name given to the role will be derived from the GitHub repository name. However, you can use the `--role-name` option to override this and set the name.

No wait

If the `--no-wait` option is present, the command will not wait for results. Results of the most recent import for any of your roles is available on the Galaxy web site by visiting *My Imports*.

Delete a role

The `delete` command requires that you first authenticate using the `login` command. Once authenticated you can remove a role from the Galaxy web site. You are only allowed to remove roles where you have access to the repository in GitHub.

Use the following to delete a role:

```
$ ansible-galaxy delete github_user github_repo
```

This only removes the role from Galaxy. It does not remove or alter the actual GitHub repository.

Travis integrations

You can create an integration or connection between a role in Galaxy and Travis. Once the connection is established, a build in Travis will automatically trigger an import in Galaxy, updating the search index with the latest information about the role.

You create the integration using the `setup` command, but before an integration can be created, you must first authenticate using the `login` command; you will also need an account in Travis, and your Travis token. Once you're ready, use the following command to create the integration:

```
$ ansible-galaxy setup travis github_user github_repo xxx-travis-token-xxx
```

The setup command requires your Travis token, however the token is not stored in Galaxy. It is used along with the GitHub username and repo to create a hash as described in the Travis documentation. The hash is stored in Galaxy and used to verify notifications received from Travis.

The setup command enables Galaxy to respond to notifications. To configure Travis to run a build on your repository and send a notification, follow the Travis getting started guide.

To instruct Travis to notify Galaxy when a build completes, add the following to your .travis.yml file:

```
notifications:
    webhooks: https://galaxy.ansible.com/api/v1/notifications/
```

List Travis integrations

Use the `--list` option to display your Travis integrations:

```
$ ansible-galaxy setup --list


ID         Source     Repo
---------- ---------- ----------
2          travis     github_user/github_repo
1          travis     github_user/github_repo
```

Remove Travis integrations

Use the `--remove` option to disable and remove a Travis integration:

```
$ ansible-galaxy setup --remove ID
```

Provide the ID of the integration to be disabled. You can find the ID by using the `--list` option.

See also:

Using Ansible collections
  Shareable collections of modules, playbooks and roles

Roles
  All about ansible roles

Mailing List
  Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels

1.26 Indexes of all modules and plugins

1.27 Playbook Keywords

These are the keywords available on common playbook objects. Keywords are one of several sources for configuring Ansible behavior. See *Controlling how Ansible behaves: precedence rules* for details on the relative precedence of each source.

Note: Please note:

- Aliases for the directives are not reflected here, nor are mutable ones. For example, *action* in task can be substituted by the name of any Ansible module.
- The keywords do not have `version_added` information at this time
- Some keywords set defaults for the objects inside of them rather than for the objects themselves

1.27.1 Play

any_errors_fatal
Force any un-handled task errors on any host to propagate to all hosts and end the play.

become
Boolean that controls if privilege escalation is used or not on *Task* execution. Implemented by the become plugin. See *Become plugins*.

become_exe
Path to the executable used to elevate privileges. Implemented by the become plugin. See *Become plugins*.

become_flags
A string of flag(s) to pass to the privilege escalation program when become is True.

become_method
Which method of privilege escalation to use (such as sudo or su).

become_user
User that you 'become' after using privilege escalation. The remote/login user must have permissions to become this user.

check_mode
A boolean that controls if a task is executed in 'check' mode. See *Validating tasks: check mode and diff mode*.

collections
List of collection namespaces to search for modules, plugins, and roles. See *Using collections in a playbook*

Note: Tasks within a role do not inherit the value of `collections` from the play. To have a role search a list of collections, use the `collections` keyword in `meta/main.yml` within a role.

connection
Allows you to change the connection plugin used for tasks to execute on the target. See *Using connection plugins*.

debugger
Enable debugging tasks based on state of the task result. See *Debugging tasks*.

diff
Toggle to make tasks return 'diff' information or not.

environment
A dictionary that gets converted into environment vars to be provided for the task upon execution. This can ONLY be used with modules. This isn't supported for any other type of plugins nor Ansible itself nor its configuration, it just sets the variables for the code responsible for executing the task. This is not a recommended way to pass in confidential data.

fact_path
Set the fact path option for the fact gathering plugin controlled by gather_facts.

force_handlers
Will force notified handler execution for hosts even if they failed during the play. Will not trigger if the play itself fails.

gather_facts
A boolean that controls if the play will automatically run the 'setup' task to gather facts for the hosts.

gather_subset
Allows you to pass subset options to the fact gathering plugin controlled by gather_facts.

gather_timeout
Allows you to set the timeout for the fact gathering plugin controlled by gather_facts.

handlers
A section with tasks that are treated as handlers, these won't get executed normally, only when notified after each section of tasks is complete. A handler's *listen* field is not templatable.

hosts
A list of groups, hosts or host pattern that translates into a list of hosts that are the play's target.

ignore_errors
Boolean that allows you to ignore task failures and continue with play. It does not affect connection errors.

ignore_unreachable
Boolean that allows you to ignore task failures due to an unreachable host and continue with the play. This does not affect other task errors (see ignore_errors) but is useful for groups of volatile/ephemeral hosts.

max_fail_percentage
can be used to abort the run after a given percentage of hosts in the current batch has failed. This only works on linear or linear derived strategies.

module_defaults
Specifies default parameter values for modules.

name
Identifier. Can be used for documentation, or in tasks/handlers.

no_log
Boolean that controls information disclosure.

order
Controls the sorting of hosts as they are used for executing the play. Possible values are inventory (default), sorted, reverse_sorted, reverse_inventory and shuffle.

port
Used to override the default port used in a connection.

post_tasks
A list of tasks to execute after the *tasks* section.

pre_tasks
A list of tasks to execute before *roles*.

remote_user
User used to log into the target via the connection plugin.

roles
List of roles to be imported into the play

run_once
Boolean that will bypass the host loop, forcing the task to attempt to execute on the first host available and afterwards apply any results and facts to all active hosts in the same batch.

serial
Explicitly define how Ansible batches the execution of the current play on the play's target. See *Setting the batch size with serial*.

strategy
Allows you to choose the connection plugin to use for the play.

tags
Tags applied to the task or included tasks, this allows selecting subsets of tasks from the command line.

tasks
Main list of tasks to execute in the play, they run after *roles* and before post_tasks.

throttle
Limit number of concurrent task runs on task, block and playbook level. This is independent of the forks and serial settings, but cannot be set higher than those limits. For example, if forks is set to 10 and the throttle is set to 15, at most 10 hosts will be operated on in parallel.

timeout
Time limit for task to execute in, if exceeded Ansible will interrupt and fail the task.

vars
Dictionary/map of variables

vars_files
List of files that contain vars to include in the play.

vars_prompt
list of variables to prompt for.

1.27.2 Role

any_errors_fatal
Force any un-handled task errors on any host to propagate to all hosts and end the play.

become
Boolean that controls if privilege escalation is used or not on *Task* execution. Implemented by the become plugin. See *Become plugins*.

become_exe
Path to the executable used to elevate privileges. Implemented by the become plugin. See *Become plugins*.

become_flags
A string of flag(s) to pass to the privilege escalation program when become is True.

become_method
Which method of privilege escalation to use (such as sudo or su).

become_user
User that you 'become' after using privilege escalation. The remote/login user must have permissions to become this user.

check_mode
A boolean that controls if a task is executed in 'check' mode. See *Validating tasks: check mode and diff mode*.

collections
List of collection namespaces to search for modules, plugins, and roles. See *Using collections in a playbook*

Note: Tasks within a role do not inherit the value of `collections` from the play. To have a role search a list of collections, use the `collections` keyword in `meta/main.yml` within a role.

connection
Allows you to change the connection plugin used for tasks to execute on the target. See *Using connection plugins*.

debugger
Enable debugging tasks based on state of the task result. See *Debugging tasks*.

delegate_facts
Boolean that allows you to apply facts to a delegated host instead of inventory_hostname.

delegate_to
Host to execute task instead of the target (inventory_hostname). Connection vars from the delegated host will also be used for the task.

diff
Toggle to make tasks return 'diff' information or not.

environment
A dictionary that gets converted into environment vars to be provided for the task upon execution. This can ONLY be used with modules. This isn't supported for any other type of plugins nor Ansible itself nor its configuration, it just sets the variables for the code responsible for executing the task. This is not a recommended way to pass in confidential data.

ignore_errors
Boolean that allows you to ignore task failures and continue with play. It does not affect connection errors.

ignore_unreachable
Boolean that allows you to ignore task failures due to an unreachable host and continue with the play. This does not affect other task errors (see ignore_errors) but is useful for groups of volatile/ephemeral hosts.

module_defaults
Specifies default parameter values for modules.

name
Identifier. Can be used for documentation, or in tasks/handlers.

no_log
Boolean that controls information disclosure.

port
Used to override the default port used in a connection.

remote_user
User used to log into the target via the connection plugin.

run_once
Boolean that will bypass the host loop, forcing the task to attempt to execute on the first host available and afterwards apply any results and facts to all active hosts in the same batch.

tags
Tags applied to the task or included tasks, this allows selecting subsets of tasks from the command line.

throttle
Limit number of concurrent task runs on task, block and playbook level. This is independent of the forks and serial settings, but cannot be set higher than those limits. For example, if forks is set to 10 and the throttle is set to 15, at most 10 hosts will be operated on in parallel.

timeout
Time limit for task to execute in, if exceeded Ansible will interrupt and fail the task.

vars
Dictionary/map of variables

when
Conditional expression, determines if an iteration of a task is run or not.

1.27.3 Block

always
List of tasks, in a block, that execute no matter if there is an error in the block or not.

any_errors_fatal
Force any un-handled task errors on any host to propagate to all hosts and end the play.

become
Boolean that controls if privilege escalation is used or not on *Task* execution. Implemented by the become plugin. See *Become plugins*.

become_exe
Path to the executable used to elevate privileges. Implemented by the become plugin. See *Become plugins*.

become_flags
A string of flag(s) to pass to the privilege escalation program when become is True.

become_method
Which method of privilege escalation to use (such as sudo or su).

become_user
User that you 'become' after using privilege escalation. The remote/login user must have permissions to become this user.

block
List of tasks in a block.

check_mode
A boolean that controls if a task is executed in 'check' mode. See *Validating tasks: check mode and diff mode*.

collections
List of collection namespaces to search for modules, plugins, and roles. See *Using collections in a playbook*

Note: Tasks within a role do not inherit the value of `collections` from the play. To have a role search a list of collections, use the `collections` keyword in `meta/main.yml` within a role.

connection
Allows you to change the connection plugin used for tasks to execute on the target. See *Using connection plugins*.

debugger
Enable debugging tasks based on state of the task result. See *Debugging tasks*.

delegate_facts
Boolean that allows you to apply facts to a delegated host instead of inventory_hostname.

delegate_to
Host to execute task instead of the target (inventory_hostname). Connection vars from the delegated host will also be used for the task.

diff
Toggle to make tasks return 'diff' information or not.

environment
A dictionary that gets converted into environment vars to be provided for the task upon execution. This can ONLY be used with modules. This isn't supported for any other type of plugins nor Ansible itself nor its configuration, it just sets the variables for the code responsible for executing the task. This is not a recommended way to pass in confidential data.

ignore_errors
Boolean that allows you to ignore task failures and continue with play. It does not affect connection errors.

ignore_unreachable
Boolean that allows you to ignore task failures due to an unreachable host and continue with the play. This does not affect other task errors (see ignore_errors) but is useful for groups of volatile/ephemeral hosts.

module_defaults
Specifies default parameter values for modules.

name
Identifier. Can be used for documentation, or in tasks/handlers.

no_log
Boolean that controls information disclosure.

notify
List of handlers to notify when the task returns a 'changed=True' status.

port
Used to override the default port used in a connection.

remote_user
User used to log into the target via the connection plugin.

rescue
List of tasks in a block that run if there is a task error in the main block list.

run_once
Boolean that will bypass the host loop, forcing the task to attempt to execute on the first host available and afterwards apply any results and facts to all active hosts in the same batch.

tags
Tags applied to the task or included tasks, this allows selecting subsets of tasks from the command line.

throttle
Limit number of concurrent task runs on task, block and playbook level. This is independent of the forks and serial settings, but cannot be set higher than those limits. For example, if forks is set to 10 and the throttle is set to 15, at most 10 hosts will be operated on in parallel.

timeout
Time limit for task to execute in, if exceeded Ansible will interrupt and fail the task.

vars
Dictionary/map of variables

when
Conditional expression, determines if an iteration of a task is run or not.

1.27.4 Task

action
The 'action' to execute for a task, it normally translates into a C(module) or action plugin.

any_errors_fatal
Force any un-handled task errors on any host to propagate to all hosts and end the play.

args
A secondary way to add arguments into a task. Takes a dictionary in which keys map to options and values.

async
Run a task asynchronously if the C(action) supports this; value is maximum runtime in seconds.

become
Boolean that controls if privilege escalation is used or not on *Task* execution. Implemented by the become plugin. See *Become plugins*.

become_exe
Path to the executable used to elevate privileges. Implemented by the become plugin. See *Become plugins*.

become_flags
A string of flag(s) to pass to the privilege escalation program when become is True.

become_method
Which method of privilege escalation to use (such as sudo or su).

become_user
User that you 'become' after using privilege escalation. The remote/login user must have permissions to become this user.

changed_when
Conditional expression that overrides the task's normal 'changed' status.

check_mode
A boolean that controls if a task is executed in 'check' mode. See *Validating tasks: check mode and diff mode*.

collections
List of collection namespaces to search for modules, plugins, and roles. See *Using collections in a playbook*

Note: Tasks within a role do not inherit the value of `collections` from the play. To have a role search a list of collections, use the `collections` keyword in `meta/main.yml` within a role.

connection
Allows you to change the connection plugin used for tasks to execute on the target. See *Using connection plugins*.

debugger
Enable debugging tasks based on state of the task result. See *Debugging tasks*.

delay
Number of seconds to delay between retries. This setting is only used in combination with until.

delegate_facts
Boolean that allows you to apply facts to a delegated host instead of inventory_hostname.

delegate_to
Host to execute task instead of the target (inventory_hostname). Connection vars from the delegated host will also be used for the task.

diff
Toggle to make tasks return 'diff' information or not.

environment
A dictionary that gets converted into environment vars to be provided for the task upon execution. This can ONLY be used with modules. This isn't supported for any other type of plugins nor Ansible itself nor its configuration, it just sets the variables for the code responsible for executing the task. This is not a recommended way to pass in confidential data.

failed_when
Conditional expression that overrides the task's normal 'failed' status.

ignore_errors
Boolean that allows you to ignore task failures and continue with play. It does not affect connection errors.

ignore_unreachable
Boolean that allows you to ignore task failures due to an unreachable host and continue with the play. This does not affect other task errors (see ignore_errors) but is useful for groups of volatile/ephemeral hosts.

local_action
Same as action but also implies `delegate_to: localhost`

loop
Takes a list for the task to iterate over, saving each list element into the `item` variable (configurable via loop_control)

loop_control
Several keys here allow you to modify/set loop behaviour in a task. See *Adding controls to loops*.

module_defaults
Specifies default parameter values for modules.

name
Identifier. Can be used for documentation, or in tasks/handlers.

no_log
Boolean that controls information disclosure.

notify
List of handlers to notify when the task returns a 'changed=True' status.

poll
Sets the polling interval in seconds for async tasks (default 10s).

port
Used to override the default port used in a connection.

register
Name of variable that will contain task status and module return data.

remote_user
User used to log into the target via the connection plugin.

retries
Number of retries before giving up in a until loop. This setting is only used in combination with until.

run_once
Boolean that will bypass the host loop, forcing the task to attempt to execute on the first host available and afterwards apply any results and facts to all active hosts in the same batch.

tags
Tags applied to the task or included tasks, this allows selecting subsets of tasks from the command line.

throttle
Limit number of concurrent task runs on task, block and playbook level. This is independent of the forks and serial settings, but cannot be set higher than those limits. For example, if forks is set to 10 and the throttle is set to 15, at most 10 hosts will be operated on in parallel.

timeout
Time limit for task to execute in, if exceeded Ansible will interrupt and fail the task.

until
This keyword implies a 'retries loop' that will go on until the condition supplied here is met or we hit the retries limit.

vars
Dictionary/map of variables

when
Conditional expression, determines if an iteration of a task is run or not.

with_<lookup_plugin>
The same as `loop` but magically adds the output of any lookup plugin to generate the item list.

1.28 Return Values

Topics

Ansible modules normally return a data structure that can be registered into a variable, or seen directly when output by the *ansible* program. Each module can optionally document its own unique return values (visible through ansible-doc and on the *main docsite*).

This document covers return values common to all modules.

Note: Some of these keys might be set by Ansible itself once it processes the module's return information.

1.28.1 Common

backup_file

For those modules that implement *backup=no|yes* when manipulating files, a path to the backup file created.

```
"backup_file": "./foo.txt.32729.2020-07-30@06:24:19~"
```

changed

A boolean indicating if the task had to make changes to the target or delegated host.

```
"changed": true
```

diff

Information on differences between the previous and current state. Often a dictionary with entries `before` and `after`, which will then be formatted by the callback plugin to a diff view.

```
"diff": [
        {
            "after": "",
            "after_header": "foo.txt (content)",
            "before": "",
            "before_header": "foo.txt (content)"
        },
        {
            "after_header": "foo.txt (file attributes)",
            "before_header": "foo.txt (file attributes)"
        }
```

failed

A boolean that indicates if the task was failed or not.

```
"failed": false
```

invocation

Information on how the module was invoked.

```
"invocation": {
        "module_args": {
            "_original_basename": "foo.txt",
            "attributes": null,
            "backup": true,
            "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
            "content": null,
            "delimiter": null,
```

(continues on next page)

(continued from previous page)

```
            "dest": "./foo.txt",
            "directory_mode": null,
            "follow": false,
            "force": true,
            "group": null,
            "local_follow": null,
            "mode": "666",
            "owner": null,
            "regexp": null,
            "remote_src": null,
            "selevel": null,
            "serole": null,
            "setype": null,
            "seuser": null,
            "src": "/Users/foo/.ansible/tmp/ansible-tmp-1596115458.110205-
↪105717464505158/source",
            "unsafe_writes": null,
            "validate": null
        }
```

msg

A string with a generic message relayed to the user.

```
"msg": "line added"
```

rc

Some modules execute command line utilities or are geared for executing commands directly (raw, shell, command, and so on), this field contains 'return code' of these utilities.

```
"rc": 257
```

results

If this key exists, it indicates that a loop was present for the task and that it contains a list of the normal module 'result' per item.

```
"results": [
    {
        "ansible_loop_var": "item",
        "backup": "foo.txt.83170.2020-07-30@07:03:05~",
        "changed": true,
        "diff": [
            {
                "after": "",
                "after_header": "foo.txt (content)",
                "before": "",
                "before_header": "foo.txt (content)"
```

(continues on next page)

(continued from previous page)

```
            },
            {
                "after_header": "foo.txt (file attributes)",
                "before_header": "foo.txt (file attributes)"
            }
        ],
        "failed": false,
        "invocation": {
            "module_args": {
                "attributes": null,
                "backrefs": false,
                "backup": true
            }
        },
        "item": "foo",
        "msg": "line added"
    },
    {
        "ansible_loop_var": "item",
        "backup": "foo.txt.83187.2020-07-30@07:03:05~",
        "changed": true,
        "diff": [
            {
                "after": "",
                "after_header": "foo.txt (content)",
                "before": "",
                "before_header": "foo.txt (content)"
            },
            {
                "after_header": "foo.txt (file attributes)",
                "before_header": "foo.txt (file attributes)"
            }
        ],
        "failed": false,
        "invocation": {
            "module_args": {
                "attributes": null,
                "backrefs": false,
                "backup": true
            }
        },
        "item": "bar",
        "msg": "line added"
    }
    ]
```

skipped

A boolean that indicates if the task was skipped or not

```
"skipped": true
```

stderr

Some modules execute command line utilities or are geared for executing commands directly (raw, shell, command, and so on), this field contains the error output of these utilities.

```
"stderr": "ls: foo: No such file or directory"
```

stderr_lines

When *stderr* is returned we also always provide this field which is a list of strings, one item per line from the original.

```
"stderr_lines": [
        "ls: doesntexist: No such file or directory"
        ]
```

stdout

Some modules execute command line utilities or are geared for executing commands directly (raw, shell, command, and so on). This field contains the normal output of these utilities.

```
"stdout": "foo!"
```

stdout_lines

When *stdout* is returned, Ansible always provides a list of strings, each containing one item per line from the original output.

```
"stdout_lines": [
"foo!"
]
```

1.28.2 Internal use

These keys can be added by modules but will be removed from registered variables; they are 'consumed' by Ansible itself.

ansible_facts

This key should contain a dictionary which will be appended to the facts assigned to the host. These will be directly accessible and don't require using a registered variable.

exception

This key can contain traceback information caused by an exception in a module. It will only be displayed on high verbosity (-vvv).

warnings

This key contains a list of strings that will be presented to the user.

deprecations

This key contains a list of dictionaries that will be presented to the user. Keys of the dictionaries are *msg* and *version*, values are string, value for the *version* key can be an empty string.

See also:

Collection Index
    Browse existing collections, modules, and plugins

GitHub modules directory
    Browse source of core and extras modules

Mailing List
    Development mailing list

Real-time chat
    How to join Ansible chat channels

1.29 Ansible Configuration Settings

Ansible supports several sources for configuring its behavior, including an ini file named `ansible.cfg`, environment variables, command-line options, playbook keywords, and variables. See *Controlling how Ansible behaves: precedence rules* for details on the relative precedence of each source.

The `ansible-config` utility allows users to see all the configuration settings available, their defaults, how to set them and where their current value comes from. See *ansible-config* for more information.

1.29.1 The configuration file

Changes can be made and used in a configuration file which will be searched for in the following order:

- `ANSIBLE_CONFIG` (environment variable if set)
- `ansible.cfg` (in the current directory)
- `~/.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg`

Ansible will process the above list and use the first file found, all others are ignored.

Note: The configuration file is one variant of an INI format. Both the hash sign (`#`) and semicolon (`;`) are allowed as comment markers when the comment starts the line. However, if the comment is inline with regular values, only the semicolon is allowed to introduce the comment. For instance:

```
# some basic default values...
inventory = /etc/ansible/hosts  ; This points to the file that lists your hosts
```

Generating a sample `ansible.cfg` file

You can generate a fully commented-out example `ansible.cfg` file, for example:

```
$ ansible-config init --disabled > ansible.cfg
```

You can also have a more complete file that includes existing plugins:

```
$ ansible-config init --disabled -t all > ansible.cfg
```

You can use these as starting points to create your own `ansible.cfg` file.

Avoiding security risks with `ansible.cfg` in the current directory

If Ansible were to load `ansible.cfg` from a world-writable current working directory, it would create a serious security risk. Another user could place their own config file there, designed to make Ansible run malicious code both locally and remotely, possibly with elevated privileges. For this reason, Ansible will not automatically load a config file from the current working directory if the directory is world-writable.

If you depend on using Ansible with a config file in the current working directory, the best way to avoid this problem is to restrict access to your Ansible directories to particular user(s) and/or group(s). If your Ansible directories live on a filesystem which has to emulate Unix permissions, like Vagrant or Windows Subsystem for Linux (WSL), you may, at first, not know how you can fix this as `chmod`, `chown`, and `chgrp` might not work there. In most of those cases, the correct fix is to modify the mount options of the filesystem so the files and directories are readable and writable by the users and groups running Ansible but closed to others. For more details on the correct settings, see:

- for Vagrant, the Vagrant documentation covers synced folder permissions.
- for WSL, the WSL docs and this Microsoft blog post cover mount options.

If you absolutely depend on storing your Ansible config in a world-writable current working directory, you can explicitly specify the config file via the *`ANSIBLE_CONFIG`* environment variable. Please take appropriate steps to mitigate the security concerns above before doing so.

Relative paths for configuration

You can specify a relative path for many configuration options. In most of those cases the path used will be relative to the `ansible.cfg` file used for the current execution. If you need a path relative to your current working directory (CWD) you can use the `{{CWD}}` macro to specify it. We do not recommend this approach, as using your CWD as the root of relative paths can be a security risk. For example: `cd /tmp; secureinfo=./newrootpassword ansible-playbook ~/safestuff/change_root_pwd.yml`.

1.29.2 Common Options

This is a copy of the options available from our release, your local install might have extra options due to additional plugins, you can use the command line utility mentioned above (*ansible-config*) to browse through those.

ACTION_WARNINGS

Description
By default Ansible will issue a warning when received from a task action (module or action plugin) These warnings can be silenced by adjusting this setting to False.

Type
boolean

Default
`True`

Version Added
2.5

Ini

Section
[defaults]

Key
action_warnings

Environment

Variable
ANSIBLE_ACTION_WARNINGS

AGNOSTIC_BECOME_PROMPT

Description
Display an agnostic become prompt instead of displaying a prompt containing the command line supplied become method

Type
boolean

Default
`True`

Version Added
2.5

Ini

Section
[privilege_escalation]

Key
agnostic_become_prompt

Environment

Variable
ANSIBLE_AGNOSTIC_BECOME_PROMPT

ANSIBLE_CONNECTION_PATH

Description
Specify where to look for the ansible-connection script. This location will be checked before searching $PATH. If null, ansible will start with the same directory as the ansible script.

Type
path

Default
`None`

Version Added
2.8

Ini

Section
[persistent_connection]

Key
ansible_connection_path

Environment

Variable
ANSIBLE_CONNECTION_PATH

ANSIBLE_COW_ACCEPTLIST

Description
Accept list of cowsay templates that are 'safe' to use, set to empty list if you want to enable all installed templates.

Type
list

Default
```
['bud-frogs', 'bunny', 'cheese', 'daemon', 'default', 'dragon',
'elephant-in-snake', 'elephant', 'eyes', 'hellokitty', 'kitty',
'luke-koala', 'meow', 'milk', 'moofasa', 'moose', 'ren', 'sheep', 'small',
'stegosaurus', 'stimpy', 'supermilker', 'three-eyes', 'turkey', 'turtle',
'tux', 'udder', 'vader-koala', 'vader', 'www']
```

Ini

Section
[defaults]

Key
cowsay_enabled_stencils :Version Added: 2.11

Environment

Variable
ANSIBLE_COW_ACCEPTLIST :Version Added: 2.11

ANSIBLE_COW_PATH

Description
Specify a custom cowsay path or swap in your cowsay implementation of choice

Type
string

Default
`None`

Ini

Section
[defaults]

Key
cowpath

Environment

Variable
ANSIBLE_COW_PATH

ANSIBLE_COW_SELECTION

Description
This allows you to chose a specific cowsay stencil for the banners or use 'random' to cycle through them.

Default
`default`

Ini

Section
[defaults]

Key
cow_selection

Environment

Variable
ANSIBLE_COW_SELECTION

ANSIBLE_FORCE_COLOR

Description
This option forces color mode even when running without a TTY or the "nocolor" setting is True.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
force_color

Environment

Variable
ANSIBLE_FORCE_COLOR

ANSIBLE_HOME

Description
The default root path for Ansible config files on the controller.

Type
path

Default
`~/.ansible`

Version Added
2.14

Ini

Section
[defaults]

Key
home

Environment

Variable
ANSIBLE_HOME

ANSIBLE_NOCOLOR

Description
: This setting allows suppressing colorizing output, which is used to give a better indication of failure and status information.

Type
: boolean

Default
: `False`

Ini

  Section
  : [defaults]

  Key
  : nocolor

Environment

- **Variable**
  ANSIBLE_NOCOLOR
- **Variable**
  NO_COLOR

  Version Added
  2.11

ANSIBLE_NOCOWS

Description
: If you have cowsay installed but want to avoid the 'cows' (why????), use this.

Type
: boolean

Default
: `False`

Ini

  Section
  : [defaults]

  Key
  : nocows

Environment

  Variable
  ANSIBLE_NOCOWS

ANSIBLE_PIPELINING

Description
This is a global option, each connection plugin can override either by having more specific options or not supporting pipelining at all. Pipelining, if supported by the connection plugin, reduces the number of network operations required to execute a module on the remote server, by executing many Ansible modules without actual file transfer. It can result in a very significant performance improvement when enabled. However this conflicts with privilege escalation (become). For example, when using 'sudo:' operations you must first disable 'requiretty' in /etc/sudoers on all managed hosts, which is why it is disabled by default. This setting will be disabled if `ANSIBLE_KEEP_REMOTE_FILES` is enabled.

Type
boolean

Default
`False`

Ini

- **Section**
  [connection]

  Key
  pipelining
- **Section**
  [defaults]

  Key
  pipelining

Environment

Variable
ANSIBLE_PIPELINING

ANY_ERRORS_FATAL

Description
Sets the default value for the any_errors_fatal keyword, if True, Task failures will be considered fatal errors.

Type
boolean

Default
`False`

Version Added
2.4

Ini

Section
[defaults]

Key
any_errors_fatal

Environment

Variable
ANSIBLE_ANY_ERRORS_FATAL

BECOME_ALLOW_SAME_USER

Description
This setting controls if become is skipped when remote user and become user are the same. I.E root sudo to root. If executable, it will be run and the resulting stdout will be used as the password.

Type
boolean

Default
`False`

Ini

Section
[privilege_escalation]

Key
become_allow_same_user

Environment

Variable
ANSIBLE_BECOME_ALLOW_SAME_USER

BECOME_PASSWORD_FILE

Description
The password file to use for the become plugin. –become-password-file. If executable, it will be run and the resulting stdout will be used as the password.

Type
path

Default
`None`

Version Added
2.12

Ini

Section
[defaults]

Key
become_password_file

Environment

Variable
ANSIBLE_BECOME_PASSWORD_FILE

BECOME_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Become Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/become:/usr/share/ansible/plugins/become" }}`

Version Added
2.8

Ini

Section
[defaults]

Key
become_plugins

Environment

Variable
ANSIBLE_BECOME_PLUGINS

CACHE_PLUGIN

Description
Chooses which cache plugin to use, the default 'memory' is ephemeral.

Default
`memory`

Ini

Section
[defaults]

Key
fact_caching

Environment

Variable
ANSIBLE_CACHE_PLUGIN

CACHE_PLUGIN_CONNECTION

Description
Defines connection or path information for the cache plugin

Default
`None`

Ini

Section
[defaults]

Key
fact_caching_connection

Environment

Variable
ANSIBLE_CACHE_PLUGIN_CONNECTION

CACHE_PLUGIN_PREFIX

Description
Prefix to use for cache plugin files/tables

Default
`ansible_facts`

Ini

Section
[defaults]

Key
fact_caching_prefix

Environment

Variable
ANSIBLE_CACHE_PLUGIN_PREFIX

CACHE_PLUGIN_TIMEOUT

Description
Expiration timeout for the cache plugin data

Type
integer

Default
`86400`

Ini

Section
[defaults]

Key
fact_caching_timeout

Environment

Variable
ANSIBLE_CACHE_PLUGIN_TIMEOUT

CALLBACKS_ENABLED

Description
List of enabled callbacks, not all callbacks need enabling, but many of those shipped with Ansible do as we don't want them activated by default.

Type
list

Default
`[]`

Ini

Section
[defaults]

Key
callbacks_enabled :Version Added: 2.11

Environment

Variable
ANSIBLE_CALLBACKS_ENABLED :Version Added: 2.11

COLLECTIONS_ON_ANSIBLE_VERSION_MISMATCH

Description
When a collection is loaded that does not support the running Ansible version (with the collection metadata key *requires_ansible*).

Default
`warning`

Choices

- **error**
  issue a 'fatal' error and stop the play
- **warning**
  issue a warning but continue
- **ignore**
  just continue silently

Ini

Section
[defaults]

Key
collections_on_ansible_version_mismatch

Environment

Variable
ANSIBLE_COLLECTIONS_ON_ANSIBLE_VERSION_MISMATCH

COLLECTIONS_PATHS

Description
Colon separated paths in which Ansible will search for collections content. Collections must be in nested *subdirectories*, not directly in these directories. For example, if `COLLECTIONS_PATHS` includes `'{{ ANSIBLE_HOME ~ "/collections" }}'`, and you want to add `my.collection` to that directory, it must be saved as `'{{ ANSIBLE_HOME} ~ "/collections/ansible_collections/my/collection" }}'`.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/collections:/usr/share/ansible/collections" }}`

Ini

- **Section**
  [defaults]

  Key
  collections_paths
- **Section**
  [defaults]

  Key
  collections_path

  Version Added
  2.10

Environment

- **Variable**
  ANSIBLE_COLLECTIONS_PATH

  Version Added
  2.10
- **Variable**
  ANSIBLE_COLLECTIONS_PATHS

COLLECTIONS_SCAN_SYS_PATH

Description
A boolean to enable or disable scanning the sys.path for installed collections

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
collections_scan_sys_path

Environment

Variable
ANSIBLE_COLLECTIONS_SCAN_SYS_PATH

COLOR_CHANGED

Description
Defines the color to use on 'Changed' task status

Default
`yellow`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
changed

Environment

Variable
ANSIBLE_COLOR_CHANGED

COLOR_CONSOLE_PROMPT

Description
Defines the default color to use for ansible-console

Default
`white`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Version Added
2.7

Ini

Section
[colors]

Key
console_prompt

Environment

Variable
ANSIBLE_COLOR_CONSOLE_PROMPT

COLOR_DEBUG

Description
Defines the color to use when emitting debug messages

Default
`dark gray`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
debug

Environment

Variable
ANSIBLE_COLOR_DEBUG

COLOR_DEPRECATE

Description
Defines the color to use when emitting deprecation messages

Default
`purple`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
deprecate

Environment

Variable
`ANSIBLE_COLOR_DEPRECATE`

COLOR_DIFF_ADD

Description
Defines the color to use when showing added lines in diffs

Default
`green`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
diff_add

Environment

Variable
ANSIBLE_COLOR_DIFF_ADD

COLOR_DIFF_LINES

Description
Defines the color to use when showing diffs

Default
`cyan`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
diff_lines

Environment

Variable
ANSIBLE_COLOR_DIFF_LINES

COLOR_DIFF_REMOVE

Description
Defines the color to use when showing removed lines in diffs

Default
`red`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
diff_remove

Environment

Variable
ANSIBLE_COLOR_DIFF_REMOVE

COLOR_ERROR

Description
Defines the color to use when emitting error messages

Default
`red`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
error

Environment

Variable
ANSIBLE_COLOR_ERROR

COLOR_HIGHLIGHT

Description
Defines the color to use for highlighting

Default
`white`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
highlight

Environment

Variable
ANSIBLE_COLOR_HIGHLIGHT

COLOR_OK

Description
Defines the color to use when showing 'OK' task status

Default
`green`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
ok

Environment

Variable
ANSIBLE_COLOR_OK

COLOR_SKIP

Description
Defines the color to use when showing 'Skipped' task status

Default
`cyan`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
skip

Environment

Variable
ANSIBLE_COLOR_SKIP

COLOR_UNREACHABLE

Description
Defines the color to use on 'Unreachable' status

Default
`bright red`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
unreachable

Environment

Variable
ANSIBLE_COLOR_UNREACHABLE

COLOR_VERBOSE

Description
Defines the color to use when emitting verbose messages. i.e those that show with '-v's.

Default
`blue`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
verbose

Environment

Variable
ANSIBLE_COLOR_VERBOSE

COLOR_WARN

Description
Defines the color to use when emitting warning messages

Default
`bright purple`

Choices

- black
- bright gray
- blue
- white
- green
- bright blue
- cyan
- bright green
- red
- bright cyan
- purple
- bright red
- yellow
- bright purple
- dark gray
- bright yellow
- magenta
- bright magenta
- normal

Ini

Section
[colors]

Key
warn

Environment

Variable
ANSIBLE_COLOR_WARN

CONNECTION_FACTS_MODULES

Description
Which modules to run during a play's fact gathering stage based on connection

Type
dict

Default
`{'asa': 'ansible.legacy.asa_facts', 'cisco.asa.asa': 'cisco.asa.asa_facts', 'eos': 'ansible.legacy.eos_facts', 'arista.eos.eos': 'arista.eos.eos_facts', 'frr': 'ansible.legacy.frr_facts', 'frr.frr.frr': 'frr.frr.frr_facts', 'ios': 'ansible.legacy.ios_facts', 'cisco.ios.ios': 'cisco.ios.ios_facts', 'iosxr': 'ansible.legacy.iosxr_facts', 'cisco.iosxr.iosxr': 'cisco.iosxr.iosxr_facts', 'junos': 'ansible.legacy.junos_facts', 'junipernetworks.junos.junos': 'junipernetworks.junos.junos_facts', 'nxos': 'ansible.legacy.nxos_facts', 'cisco.nxos.nxos': 'cisco.nxos.nxos_facts', 'vyos': 'ansible.legacy.vyos_facts', 'vyos.vyos.vyos': 'vyos.vyos.vyos_facts', 'exos': 'ansible.legacy.exos_facts', 'extreme.exos.exos': 'extreme.exos.exos_facts', 'slxos': 'ansible.legacy.slxos_facts', 'extreme.slxos.slxos': 'extreme.slxos.slxos_facts', 'voss': 'ansible.legacy.voss_facts', 'extreme.voss.voss': 'extreme.voss.voss_facts', 'ironware': 'ansible.legacy.ironware_facts', 'community.network.ironware': 'community.network.ironware_facts'}`

CONNECTION_PASSWORD_FILE

Description
The password file to use for the connection plugin. –connection-password-file.

Type
path

Default
`None`

Version Added
2.12

Ini

Section
[defaults]

Key
connection_password_file

Environment

Variable
ANSIBLE_CONNECTION_PASSWORD_FILE

COVERAGE_REMOTE_OUTPUT

Description
Sets the output directory on the remote host to generate coverage reports to. Currently only used for remote coverage on PowerShell modules. This is for internal use only.

Type
str

Version Added
2.9

Environment

Variable
_ANSIBLE_COVERAGE_REMOTE_OUTPUT

Variables

name
_ansible_coverage_remote_output

COVERAGE_REMOTE_PATHS

Description
A list of paths for files on the Ansible controller to run coverage for when executing on the remote host. Only files that match the path glob will have its coverage collected. Multiple path globs can be specified and are separated by `:`. Currently only used for remote coverage on PowerShell modules. This is for internal use only.

Type
str

Default
*

Version Added
2.9

Environment

Variable
_ANSIBLE_COVERAGE_REMOTE_PATH_FILTER

DEFAULT_ACTION_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Action Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/action:/usr/share/ansible/plugins/action" }}`

Ini

Section
[defaults]

Key
action_plugins

Environment

Variable
ANSIBLE_ACTION_PLUGINS

DEFAULT_ALLOW_UNSAFE_LOOKUPS

Description
When enabled, this option allows lookup plugins (whether used in variables as `{{lookup('foo')}}` or as a loop as with_foo) to return data that is not marked 'unsafe'. By default, such data is marked as unsafe to prevent the templating engine from evaluating any jinja2 templating language, as this could represent a security risk. This option is provided to allow for backward compatibility, however users should first consider adding allow_unsafe=True to any lookups which may be expected to contain data which may be run through the templating engine late

Type
boolean

Default
`False`

Version Added
2.2.3

Ini

Section
[defaults]

Key
allow_unsafe_lookups

DEFAULT_ASK_PASS

Description
This controls whether an Ansible playbook should prompt for a login password. If using SSH keys for authentication, you probably do not need to change this setting.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
ask_pass

Environment

Variable
ANSIBLE_ASK_PASS

DEFAULT_ASK_VAULT_PASS

Description
This controls whether an Ansible playbook should prompt for a vault password.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
ask_vault_pass

Environment

Variable
ANSIBLE_ASK_VAULT_PASS

DEFAULT_BECOME

Description
Toggles the use of privilege escalation, allowing you to 'become' another user after login.

Type
boolean

Default
`False`

Ini

Section
[privilege_escalation]

Key
become

Environment

Variable
ANSIBLE_BECOME

DEFAULT_BECOME_ASK_PASS

Description
Toggle to prompt for privilege escalation password.

Type
boolean

Default
`False`

Ini

Section
[privilege_escalation]

Key
become_ask_pass

Environment

Variable
ANSIBLE_BECOME_ASK_PASS

DEFAULT_BECOME_EXE

Description
executable to use for privilege escalation, otherwise Ansible will depend on PATH

Default
`None`

Ini

Section
[privilege_escalation]

Key
become_exe

Environment

Variable
ANSIBLE_BECOME_EXE

DEFAULT_BECOME_FLAGS

Description
Flags to pass to the privilege escalation executable.

Default

Ini

Section
[privilege_escalation]

Key
become_flags

Environment

Variable
ANSIBLE_BECOME_FLAGS

DEFAULT_BECOME_METHOD

Description
Privilege escalation method to use when *become* is enabled.

Default
`sudo`

Ini

Section
[privilege_escalation]

Key
become_method

Environment

Variable
ANSIBLE_BECOME_METHOD

DEFAULT_BECOME_USER

Description
The user your login/remote user 'becomes' when using privilege escalation, most systems will use 'root' when no user is specified.

Default
`root`

Ini

Section
[privilege_escalation]

Key
become_user

Environment

Variable
ANSIBLE_BECOME_USER

DEFAULT_CACHE_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Cache Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/cache:/usr/share/ansible/plugins/cache" }}`

Ini

Section
[defaults]

Key
cache_plugins

Environment

Variable
ANSIBLE_CACHE_PLUGINS

DEFAULT_CALLBACK_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Callback Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/callback:/usr/share/ansible/plugins/callback" }}`

Ini

Section
[defaults]

Key
callback_plugins

Environment

Variable
ANSIBLE_CALLBACK_PLUGINS

DEFAULT_CLICONF_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Cliconf Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/cliconf:/usr/share/ansible/plugins/cliconf" }}`

Ini

Section
[defaults]

Key
cliconf_plugins

Environment

Variable
ANSIBLE_CLICONF_PLUGINS

DEFAULT_CONNECTION_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Connection Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/connection:/usr/share/ansible/plugins/connection" }}`

Ini

Section
[defaults]

Key
connection_plugins

Environment

Variable
ANSIBLE_CONNECTION_PLUGINS

DEFAULT_DEBUG

Description
Toggles debug output in Ansible. This is *very* verbose and can hinder multiprocessing. Debug output can also include secret information despite no_log settings being enabled, which means debug mode should not be used in production.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
debug

Environment

Variable
ANSIBLE_DEBUG

DEFAULT_EXECUTABLE

Description
This indicates the command to use to spawn a shell under for Ansible's execution needs on a target. Users may need to change this in rare instances when shell usage is constrained, but in most cases it may be left as is.

Default
`/bin/sh`

Ini

Section
[defaults]

Key
executable

Environment

Variable
ANSIBLE_EXECUTABLE

DEFAULT_FACT_PATH

Description
This option allows you to globally configure a custom path for 'local_facts' for the implied ansible_collections.ansible.builtin.setup_module task when using fact gathering. If not set, it will fallback to the default from the `ansible.builtin.setup` module: `/etc/ansible/facts.d`. This does **not** affect user defined tasks that use the `ansible.builtin.setup` module. The real action being created by the implicit task is currently `ansible.legacy.gather_facts` module, which then calls the configured fact modules, by default this will be `ansible.builtin.setup` for POSIX systems but other platforms might have different defaults.

Type
string

Ini

Section
[defaults]

Key
fact_path

Environment

Variable
ANSIBLE_FACT_PATH

Deprecated in
2.18

Deprecated detail
the module_defaults keyword is a more generic version and can apply to all calls to the M(ansible.builtin.gather_facts) or M(ansible.builtin.setup) actions

Deprecated alternatives
module_defaults

DEFAULT_FILTER_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Jinja2 Filter Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/filter:/usr/share/ansible/plugins/filter" }}`

Ini

Section
[defaults]

Key
filter_plugins

Environment

Variable
ANSIBLE_FILTER_PLUGINS

DEFAULT_FORCE_HANDLERS

Description
This option controls if notified handlers run on a host even if a failure occurs on that host. When false, the handlers will not run if a failure has occurred on a host. This can also be set per play or on the command line. See Handlers and Failure for more details.

Type
boolean

Default
`False`

Version Added
1.9.1

Ini

Section
[defaults]

Key
force_handlers

Environment

Variable
ANSIBLE_FORCE_HANDLERS

DEFAULT_FORKS

Description
Maximum number of forks Ansible will use to execute tasks on target hosts.

Type
integer

Default
5

Ini

Section
[defaults]

Key
forks

Environment

Variable
ANSIBLE_FORKS

DEFAULT_GATHER_SUBSET

Description
Set the *gather_subset* option for the ansible_collections.ansible.builtin.setup_module task in the implicit fact gathering. See the module documentation for specifics. It does **not** apply to user defined `ansible.builtin.setup` tasks.

Type
list

Version Added
2.1

Ini

Section
[defaults]

Key
gather_subset

Environment

Variable
ANSIBLE_GATHER_SUBSET

Deprecated in
2.18

Deprecated detail
the module_defaults keyword is a more generic version and can apply to all calls to the M(ansible.builtin.gather_facts) or M(ansible.builtin.setup) actions

Deprecated alternatives
module_defaults

DEFAULT_GATHER_TIMEOUT

Description
Set the timeout in seconds for the implicit fact gathering, see the module documentation for specifics. It does **not** apply to user defined ansible_collections.ansible.builtin.setup_module tasks.

Type
integer

Ini

Section
[defaults]

Key
gather_timeout

Environment

Variable
ANSIBLE_GATHER_TIMEOUT

Deprecated in
2.18

Deprecated detail
the module_defaults keyword is a more generic version and can apply to all calls to the M(ansible.builtin.gather_facts) or M(ansible.builtin.setup) actions

Deprecated alternatives
module_defaults

DEFAULT_GATHERING

Description
This setting controls the default policy of fact gathering (facts discovered about remote systems). This option can be useful for those wishing to save fact gathering time. Both 'smart' and 'explicit' will use the cache plugin.

Default
`implicit`

Choices

- **implicit**
  the cache plugin will be ignored and facts will be gathered per play unless 'gather_facts: False' is set.
- **explicit**
  facts will not be gathered unless directly requested in the play.
- **smart**
  each new host that has no facts discovered will be scanned, but if the same host is addressed in multiple plays it will not be contacted again in the run.

Version Added
1.6

Ini

Section
[defaults]

Key
gathering

Environment

Variable
ANSIBLE_GATHERING

DEFAULT_HASH_BEHAVIOUR

Description
This setting controls how duplicate definitions of dictionary variables (aka hash, map, associative array) are handled in Ansible. This does not affect variables whose values are scalars (integers, strings) or arrays. **WARNING**, changing this setting is not recommended as this is fragile and makes your content (plays, roles, collections) non portable, leading to continual confusion and misuse. Don't change this setting unless you think you have an absolute need for it. We recommend avoiding reusing variable names and relying on the `combine` filter and `vars` and `varnames` lookups to create merged versions of the individual variables. In our experience this is rarely really needed and a sign that too much complexity has been introduced into the data structures and plays. For some uses you can also look into custom vars_plugins to merge on input, even substituting the default `host_group_vars` that is in charge of parsing the `host_vars/` and `group_vars/` directories. Most users of this setting are only interested in inventory scope, but the setting itself affects all sources and makes debugging even harder. All playbooks and roles in the official examples repos assume the default for this setting. Changing the setting to `merge` applies across variable sources, but many sources will internally still overwrite the variables. For example `include_vars` will dedupe variables internally before updating Ansible, with 'last defined' overwriting previous definitions in same file. The Ansible project recommends you **avoid ``merge`` for new projects.** It is the intention of the Ansible developers to eventually deprecate and remove this setting, but it is being kept as some users do heavily rely on it. New projects should **avoid 'merge'**.

Type
string

Default
`replace`

Choices

- **replace**
  Any variable that is defined more than once is overwritten using the order from variable precedence rules (highest wins).
- **merge**
  Any dictionary variable will be recursively merged with new definitions across the different variable definition sources.

Ini

Section
[defaults]

Key
hash_behaviour

Environment

Variable
ANSIBLE_HASH_BEHAVIOUR

DEFAULT_HOST_LIST

Description
Comma separated list of Ansible inventory sources

Type
pathlist

Default
`/etc/ansible/hosts`

Ini

Section
[defaults]

Key
inventory

Environment

Variable
ANSIBLE_INVENTORY

DEFAULT_HTTPAPI_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for HttpApi Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/httpapi:/usr/share/ansible/plugins/httpapi" }}`

Ini

Section
[defaults]

Key
httpapi_plugins

Environment

Variable
ANSIBLE_HTTPAPI_PLUGINS

DEFAULT_INTERNAL_POLL_INTERVAL

Description
This sets the interval (in seconds) of Ansible internal processes polling each other. Lower values improve performance with large playbooks at the expense of extra CPU load. Higher values are more suitable for Ansible usage in automation scenarios, when UI responsiveness is not required but CPU usage might be a concern. The default corresponds to the value hardcoded in Ansible <= 2.1

Type
float

Default
`0.001`

Version Added
2.2

Ini

Section
[defaults]

Key
internal_poll_interval

DEFAULT_INVENTORY_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Inventory Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/inventory:/usr/share/ansible/plugins/inventory" }}`

Ini

Section
[defaults]

Key
inventory_plugins

Environment

Variable
ANSIBLE_INVENTORY_PLUGINS

DEFAULT_JINJA2_EXTENSIONS

Description
This is a developer-specific feature that allows enabling additional Jinja2 extensions. See the Jinja2 documentation for details. If you do not know what these do, you probably don't need to change this setting :)

Default
`[]`

Ini

Section
[defaults]

Key
jinja2_extensions

Environment

Variable
ANSIBLE_JINJA2_EXTENSIONS

DEFAULT_JINJA2_NATIVE

Description
This option preserves variable types during template operations.

Type
boolean

Default
`False`

Version Added
2.7

Ini

Section
[defaults]

Key
jinja2_native

Environment

Variable
ANSIBLE_JINJA2_NATIVE

DEFAULT_KEEP_REMOTE_FILES

Description
Enables/disables the cleaning up of the temporary files Ansible used to execute the tasks on the remote. If this option is enabled it will disable `ANSIBLE_PIPELINING`.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
keep_remote_files

Environment

Variable
ANSIBLE_KEEP_REMOTE_FILES

DEFAULT_LIBVIRT_LXC_NOSECLABEL

Description
This setting causes libvirt to connect to lxc containers by passing –noseclabel to virsh. This is necessary when running on systems which do not have SELinux.

Type
boolean

Default
`False`

Version Added
2.1

Ini

Section
[selinux]

Key
libvirt_lxc_noseclabel

Environment

Variable
ANSIBLE_LIBVIRT_LXC_NOSECLABEL

DEFAULT_LOAD_CALLBACK_PLUGINS

Description
Controls whether callback plugins are loaded when running /usr/bin/ansible. This may be used to log activity from the command line, send notifications, and so on. Callback plugins are always loaded for `ansible-playbook`.

Type
boolean

Default
`False`

Version Added
1.8

Ini

Section
[defaults]

Key
bin_ansible_callbacks

Environment

Variable
ANSIBLE_LOAD_CALLBACK_PLUGINS

DEFAULT_LOCAL_TMP

Description
Temporary directory for Ansible to use on the controller.

Type
tmppath

Default
`{{ ANSIBLE_HOME ~ "/tmp" }}`

Ini

Section
[defaults]

Key
local_tmp

Environment

Variable
ANSIBLE_LOCAL_TEMP

DEFAULT_LOG_FILTER

Description
List of logger names to filter out of the log file

Type
list

Default
`[]`

Ini

Section
[defaults]

Key
log_filter

Environment

Variable
ANSIBLE_LOG_FILTER

DEFAULT_LOG_PATH

Description
File to which Ansible will log on the controller. When empty logging is disabled.

Type
path

Default
`None`

Ini

Section
[defaults]

Key
log_path

Environment

Variable
ANSIBLE_LOG_PATH

DEFAULT_LOOKUP_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Lookup Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/lookup:/usr/share/ansible/plugins/lookup" }}`

Ini

Section
[defaults]

Key
lookup_plugins

Environment

Variable
ANSIBLE_LOOKUP_PLUGINS

DEFAULT_MANAGED_STR

Description
Sets the macro for the 'ansible_managed' variable available for ansible_collections.ansible.builtin.template_module and ansible_collections.ansible.windows.win_template_module. This is only relevant for those two modules.

Default
`Ansible managed`

Ini

Section
[defaults]

Key
ansible_managed

DEFAULT_MODULE_ARGS

Description
This sets the default arguments to pass to the `ansible` adhoc binary if no `-a` is specified.

Default
`None`

Ini

Section
[defaults]

Key
module_args

Environment

Variable
ANSIBLE_MODULE_ARGS

DEFAULT_MODULE_COMPRESSION

Description
Compression scheme to use when transferring Python modules to the target.

Default
`ZIP_DEFLATED`

Ini

Section
[defaults]

Key
module_compression

DEFAULT_MODULE_NAME

Description
Module to use with the `ansible` AdHoc command, if none is specified via `-m`.

Default
`command`

Ini

Section
[defaults]

Key
module_name

DEFAULT_MODULE_PATH

Description
Colon separated paths in which Ansible will search for Modules.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/modules:/usr/share/ansible/plugins/modules" }}`

Ini

Section
[defaults]

Key
library

Environment

Variable
ANSIBLE_LIBRARY

DEFAULT_MODULE_UTILS_PATH

Description
Colon separated paths in which Ansible will search for Module utils files, which are shared by modules.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/module_utils:/usr/share/ansible/plugins/module_utils" }}`

Ini

Section
[defaults]

Key
module_utils

Environment

Variable
ANSIBLE_MODULE_UTILS

DEFAULT_NETCONF_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Netconf Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/netconf:/usr/share/ansible/plugins/netconf" }}`

Ini

Section
[defaults]

Key
netconf_plugins

Environment

Variable
ANSIBLE_NETCONF_PLUGINS

DEFAULT_NO_LOG

Description
Toggle Ansible's display and logging of task details, mainly used to avoid security disclosures.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
no_log

Environment

Variable
ANSIBLE_NO_LOG

DEFAULT_NO_TARGET_SYSLOG

Description
Toggle Ansible logging to syslog on the target when it executes tasks. On Windows hosts this will disable a newer style PowerShell modules from writing to the event log.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
no_target_syslog

Environment

Variable
ANSIBLE_NO_TARGET_SYSLOG

Variables

name
ansible_no_target_syslog :Version Added: 2.10

DEFAULT_NULL_REPRESENTATION

Description
What templating should return as a 'null' value. When not set it will let Jinja2 decide.

Type
raw

Default
`None`

Ini

Section
[defaults]

Key
null_representation

Environment

Variable
ANSIBLE_NULL_REPRESENTATION

DEFAULT_POLL_INTERVAL

Description
For asynchronous tasks in Ansible (covered in Asynchronous Actions and Polling), this is how often to check back on the status of those tasks when an explicit poll interval is not supplied. The default is a reasonably moderate 15 seconds which is a tradeoff between checking in frequently and providing a quick turnaround when something may have completed.

Type
integer

Default
`15`

Ini

Section
[defaults]

Key
poll_interval

Environment

Variable
ANSIBLE_POLL_INTERVAL

DEFAULT_PRIVATE_KEY_FILE

Description
Option for connections using a certificate or key file to authenticate, rather than an agent or passwords, you can set the default value here to avoid re-specifying –private-key with every invocation.

Type
path

Default
`None`

Ini

Section
[defaults]

Key
private_key_file

Environment

Variable
ANSIBLE_PRIVATE_KEY_FILE

DEFAULT_PRIVATE_ROLE_VARS

Description
Makes role variables inaccessible from other roles. This was introduced as a way to reset role variables to default values if a role is used more than once in a playbook.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
private_role_vars

Environment

Variable
ANSIBLE_PRIVATE_ROLE_VARS

DEFAULT_REMOTE_PORT

Description
Port to use in remote connections, when blank it will use the connection plugin default.

Type
integer

Default
`None`

Ini

Section
[defaults]

Key
remote_port

Environment

Variable
ANSIBLE_REMOTE_PORT

DEFAULT_REMOTE_USER

Description
Sets the login user for the target machines When blank it uses the connection plugin's default, normally the user currently executing Ansible.

Ini

Section
[defaults]

Key
remote_user

Environment

Variable
ANSIBLE_REMOTE_USER

DEFAULT_ROLES_PATH

Description
Colon separated paths in which Ansible will search for Roles.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/roles:/usr/share/ansible/roles:/etc/ansible/roles" }}`

Ini

Section
[defaults]

Key
roles_path

Environment

Variable
ANSIBLE_ROLES_PATH

DEFAULT_SELINUX_SPECIAL_FS

Description
Some filesystems do not support safe operations and/or return inconsistent errors, this setting makes Ansible 'tolerate' those in the list w/o causing fatal errors. Data corruption may occur and writes are not always verified when a filesystem is in the list.

Type
list

Default
`fuse, nfs, vboxsf, ramfs, 9p, vfat`

Ini

Section
[selinux]

Key
special_context_filesystems

Environment

Variable
ANSIBLE_SELINUX_SPECIAL_FS :Version Added: 2.9

DEFAULT_STDOUT_CALLBACK

Description
Set the main callback used to display Ansible output. You can only have one at a time. You can have many other callbacks, but just one can be in charge of stdout. See *Callback plugins* for a list of available options.

Default
`default`

Ini

Section
[defaults]

Key
stdout_callback

Environment

Variable
ANSIBLE_STDOUT_CALLBACK

DEFAULT_STRATEGY

Description
Set the default strategy used for plays.

Default
`linear`

Version Added
2.3

Ini

Section
[defaults]

Key
strategy

Environment

Variable
ANSIBLE_STRATEGY

DEFAULT_STRATEGY_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Strategy Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/strategy:/usr/share/ansible/plugins/strategy" }}`

Ini

Section
[defaults]

Key
strategy_plugins

Environment

Variable
ANSIBLE_STRATEGY_PLUGINS

DEFAULT_SU

Description
Toggle the use of “su” for tasks.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
su

Environment

Variable
ANSIBLE_SU

DEFAULT_SYSLOG_FACILITY

Description
Syslog facility to use when Ansible logs to the remote target

Default
`LOG_USER`

Ini

Section
[defaults]

Key
syslog_facility

Environment

Variable
ANSIBLE_SYSLOG_FACILITY

DEFAULT_TERMINAL_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Terminal Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/terminal:/usr/share/ansible/plugins/terminal" }}`

Ini

Section
[defaults]

Key
terminal_plugins

Environment

Variable
ANSIBLE_TERMINAL_PLUGINS

DEFAULT_TEST_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Jinja2 Test Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/test:/usr/share/ansible/plugins/test" }}`

Ini

Section
[defaults]

Key
test_plugins

Environment

Variable
ANSIBLE_TEST_PLUGINS

DEFAULT_TIMEOUT

Description
This is the default timeout for connection plugins to use.

Type
integer

Default
`10`

Ini

Section
[defaults]

Key
timeout

Environment

Variable
ANSIBLE_TIMEOUT

DEFAULT_TRANSPORT

Description
Default connection plugin to use, the 'smart' option will toggle between 'ssh' and 'paramiko' depending on controller OS and ssh versions

Default
`smart`

Ini

Section
[defaults]

Key
transport

Environment

Variable
ANSIBLE_TRANSPORT

DEFAULT_UNDEFINED_VAR_BEHAVIOR

Description
When True, this causes ansible templating to fail steps that reference variable names that are likely typoed. Otherwise, any '{{ template_expression }}' that contains undefined variables will be rendered in a template or ansible action line exactly as written.

Type
boolean

Default
`True`

Version Added
1.3

Ini

Section
[defaults]

Key
error_on_undefined_vars

Environment

Variable
ANSIBLE_ERROR_ON_UNDEFINED_VARS

DEFAULT_VARS_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Vars Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/vars:/usr/share/ansible/plugins/vars" }}`

Ini

Section
[defaults]

Key
vars_plugins

Environment

Variable
ANSIBLE_VARS_PLUGINS

DEFAULT_VAULT_ENCRYPT_IDENTITY

Description
The vault_id to use for encrypting by default. If multiple vault_ids are provided, this specifies which to use for encryption. The –encrypt-vault-id cli option overrides the configured value.

Ini

Section
[defaults]

Key
vault_encrypt_identity

Environment

Variable
ANSIBLE_VAULT_ENCRYPT_IDENTITY

DEFAULT_VAULT_ID_MATCH

Description
If true, decrypting vaults with a vault id will only try the password from the matching vault-id

Default
`False`

Ini

Section
[defaults]

Key
vault_id_match

Environment

Variable
ANSIBLE_VAULT_ID_MATCH

DEFAULT_VAULT_IDENTITY

Description
The label to use for the default vault id label in cases where a vault id label is not provided

Default
`default`

Ini

Section
[defaults]

Key
vault_identity

Environment

Variable
ANSIBLE_VAULT_IDENTITY

DEFAULT_VAULT_IDENTITY_LIST

Description
A list of vault-ids to use by default. Equivalent to multiple –vault-id args. Vault-ids are tried in order.

Type
list

Default
`[]`

Ini

Section
[defaults]

Key
vault_identity_list

Environment

Variable
ANSIBLE_VAULT_IDENTITY_LIST

DEFAULT_VAULT_PASSWORD_FILE

Description
The vault password file to use. Equivalent to –vault-password-file or –vault-id If executable, it will be run and the resulting stdout will be used as the password.

Type
path

Default
`None`

Ini

Section
[defaults]

Key
vault_password_file

Environment

Variable
ANSIBLE_VAULT_PASSWORD_FILE

DEFAULT_VERBOSITY

Description
Sets the default verbosity, equivalent to the number of `-v` passed in the command line.

Type
integer

Default
`0`

Ini

Section
[defaults]

Key
verbosity

Environment

Variable
ANSIBLE_VERBOSITY

DEPRECATION_WARNINGS

Description
Toggle to control the showing of deprecation warnings

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
deprecation_warnings

Environment

Variable
ANSIBLE_DEPRECATION_WARNINGS

DEVEL_WARNING

Description
Toggle to control showing warnings related to running devel

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
devel_warning

Environment

Variable
ANSIBLE_DEVEL_WARNING

DIFF_ALWAYS

Description
Configuration toggle to tell modules to show differences when in 'changed' status, equivalent to `--diff`.

Type
bool

Default
`False`

Ini

Section
[diff]

Key
always

Environment

Variable
ANSIBLE_DIFF_ALWAYS

DIFF_CONTEXT

Description
How many lines of context to show when displaying the differences between files.

Type
integer

Default
`3`

Ini

Section
[diff]

Key
context

Environment

Variable
ANSIBLE_DIFF_CONTEXT

DISPLAY_ARGS_TO_STDOUT

Description
Normally `ansible-playbook` will print a header for each task that is run. These headers will contain the name: field from the task if you specified one. If you didn't then `ansible-playbook` uses the task's action to help you tell which task is presently running. Sometimes you run many of the same action and so you want more information about the task to differentiate it from others of the same action. If you set this variable to True in the config then `ansible-playbook` will also include the task's arguments in the header. This setting defaults to False because there is a chance that you have sensitive values in your parameters and you do not want those to be printed. If you set this to True you should be sure that you have secured your environment's stdout (no one can shoulder surf your screen and you aren't saving stdout to an insecure file) or made sure that all of your playbooks explicitly added the `no_log: True` parameter to tasks which have sensitive values See How do I keep secret data in my playbook? for more information.

Type
boolean

Default
`False`

Version Added
2.1

Ini

Section
[defaults]

Key
display_args_to_stdout

Environment

Variable
ANSIBLE_DISPLAY_ARGS_TO_STDOUT

DISPLAY_SKIPPED_HOSTS

Description
Toggle to control displaying skipped task/host entries in a task in the default callback

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
display_skipped_hosts

Environment

Variable
ANSIBLE_DISPLAY_SKIPPED_HOSTS

DOC_FRAGMENT_PLUGIN_PATH

Description
Colon separated paths in which Ansible will search for Documentation Fragments Plugins.

Type
pathspec

Default
`{{ ANSIBLE_HOME ~ "/plugins/doc_fragments:/usr/share/ansible/plugins/doc_fragments" }}`

Ini

Section
[defaults]

Key
doc_fragment_plugins

Environment

Variable
ANSIBLE_DOC_FRAGMENT_PLUGINS

DOCSITE_ROOT_URL

Description
Root docsite URL used to generate docs URLs in warning/error text; must be an absolute URL with valid scheme and trailing slash.

Default
`https://docs.ansible.com/ansible-core/`

Version Added
2.8

Ini

Section
[defaults]

Key
docsite_root_url

DUPLICATE_YAML_DICT_KEY

Description
By default Ansible will issue a warning when a duplicate dict key is encountered in YAML. These warnings can be silenced by adjusting this setting to False.

Type
string

Default
`warn`

Choices

- **error**
  issue a ‘fatal’ error and stop the play

- **warn**
  issue a warning but continue
- **ignore**
  just continue silently

Version Added
2.9

Ini

Section
[defaults]

Key
duplicate_dict_key

Environment

Variable
ANSIBLE_DUPLICATE_YAML_DICT_KEY

EDITOR

Default
`vi`

Ini

Section
[defaults]

Key
editor :Version Added: 2.15

Environment

- **Variable**
  ANSIBLE_EDITOR

  Version Added
  2.15
- **Variable**
  EDITOR

ENABLE_TASK_DEBUGGER

Description
Whether or not to enable the task debugger, this previously was done as a strategy plugin. Now all strategy plugins can inherit this behavior. The debugger defaults to activating when a task is failed on unreachable. Use the debugger keyword for more flexibility.

Type
boolean

Default
`False`

Version Added
2.5

Ini

Section
[defaults]

Key
enable_task_debugger

Environment

Variable
ANSIBLE_ENABLE_TASK_DEBUGGER

ERROR_ON_MISSING_HANDLER

Description
Toggle to allow missing handlers to become a warning instead of an error when notifying.

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
error_on_missing_handler

Environment

Variable
ANSIBLE_ERROR_ON_MISSING_HANDLER

FACTS_MODULES

Description
Which modules to run during a play's fact gathering stage, using the default of 'smart' will try to figure it out based on connection type. If adding your own modules but you still want to use the default Ansible facts, you will want to include 'setup' or corresponding network module to the list (if you add 'smart', Ansible will also figure it out). This does not affect explicit calls to the 'setup' module, but does always affect the 'gather_facts' action (implicit or explicit).

Type
list

Default
`['smart']`

Ini

Section
[defaults]

Key
facts_modules

Environment

Variable
ANSIBLE_FACTS_MODULES

Variables

name
ansible_facts_modules

GALAXY_CACHE_DIR

Description
The directory that stores cached responses from a Galaxy server. This is only used by the `ansible-galaxy collection install` and `download` commands. Cache files inside this dir will be ignored if they are world writable.

Type
path

Default
`{{ ANSIBLE_HOME ~ "/galaxy_cache" }}`

Version Added
2.11

Ini

Section
[galaxy]

Key
cache_dir

Environment

Variable
ANSIBLE_GALAXY_CACHE_DIR

GALAXY_COLLECTION_SKELETON

Description
Collection skeleton directory to use as a template for the `init` action in `ansible-galaxy collection`, same as `--collection-skeleton`.

Type
path

Ini

Section
[galaxy]

Key
collection_skeleton

Environment

Variable
ANSIBLE_GALAXY_COLLECTION_SKELETON

GALAXY_COLLECTION_SKELETON_IGNORE

Description
patterns of files to ignore inside a Galaxy collection skeleton directory

Type
list

Default
`['^.git$', '^.*/.git_keep$']`

Ini

Section
[galaxy]

Key
collection_skeleton_ignore

Environment

Variable
ANSIBLE_GALAXY_COLLECTION_SKELETON_IGNORE

GALAXY_DISABLE_GPG_VERIFY

Description
Disable GPG signature verification during collection installation.

Type
bool

Default
`False`

Version Added
2.13

Ini

Section
[galaxy]

Key
disable_gpg_verify

Environment

Variable
ANSIBLE_GALAXY_DISABLE_GPG_VERIFY

GALAXY_DISPLAY_PROGRESS

Description
Some steps in `ansible-galaxy` display a progress wheel which can cause issues on certain displays or when outputting the stdout to a file. This config option controls whether the display wheel is shown or not. The default is to show the display wheel if stdout has a tty.

Type
bool

Default
`None`

Version Added
2.10

Ini

Section
[galaxy]

Key
display_progress

Environment

Variable
ANSIBLE_GALAXY_DISPLAY_PROGRESS

GALAXY_GPG_KEYRING

Description
Configure the keyring used for GPG signature verification during collection installation and verification.

Type
path

Version Added
2.13

Ini

Section
[galaxy]

Key
gpg_keyring

Environment

Variable
ANSIBLE_GALAXY_GPG_KEYRING

GALAXY_IGNORE_CERTS

Description
If set to yes, ansible-galaxy will not validate TLS certificates. This can be useful for testing against a server with a self-signed certificate.

Type
boolean

Ini

Section
[galaxy]

Key
ignore_certs

Environment

Variable
ANSIBLE_GALAXY_IGNORE

GALAXY_IGNORE_INVALID_SIGNATURE_STATUS_CODES

Description
A list of GPG status codes to ignore during GPG signature verification. See L(https://github.com/gpg/gnupg/blob/master/doc/DETAILS#general-status-codes) for status code descriptions. If fewer signatures successfully verify the collection than *GALAXY_REQUIRED_VALID_SIGNATURE_COUNT*, signature verification will fail even if all error codes are ignored.

Type
list

Choices

- EXPSIG
- EXPKEYSIG
- REVKEYSIG
- BADSIG
- ERRSIG
- NO_PUBKEY
- MISSING_PASSPHRASE
- BAD_PASSPHRASE
- NODATA
- UNEXPECTED
- ERROR
- FAILURE
- BADARMOR
- KEYEXPIRED
- KEYREVOKED

- NO_SECKEY

Ini

Section
[galaxy]

Key
ignore_signature_status_codes

Environment

Variable
ANSIBLE_GALAXY_IGNORE_SIGNATURE_STATUS_CODES

GALAXY_REQUIRED_VALID_SIGNATURE_COUNT

Description
The number of signatures that must be successful during GPG signature verification while installing or verifying collections. This should be a positive integer or all to indicate all signatures must successfully validate the collection. Prepend + to the value to fail if no valid signatures are found for the collection.

Type
str

Default
1

Ini

Section
[galaxy]

Key
required_valid_signature_count

Environment

Variable
ANSIBLE_GALAXY_REQUIRED_VALID_SIGNATURE_COUNT

GALAXY_ROLE_SKELETON

Description
Role skeleton directory to use as a template for the `init` action in `ansible-galaxy`/`ansible-galaxy role`, same as `--role-skeleton`.

Type
path

Ini

Section
[galaxy]

Key
role_skeleton

Environment

Variable
ANSIBLE_GALAXY_ROLE_SKELETON

GALAXY_ROLE_SKELETON_IGNORE

Description
patterns of files to ignore inside a Galaxy role or collection skeleton directory

Type
list

Default
`['^.git$', '^.*/.git_keep$']`

Ini

Section
[galaxy]

Key
role_skeleton_ignore

Environment

Variable
ANSIBLE_GALAXY_ROLE_SKELETON_IGNORE

GALAXY_SERVER

Description
URL to prepend when roles don't specify the full URI, assume they are referencing this server as the source.

Default
`https://galaxy.ansible.com`

Ini

Section
[galaxy]

Key
server

Environment

Variable
ANSIBLE_GALAXY_SERVER

GALAXY_SERVER_LIST

Description
A list of Galaxy servers to use when installing a collection. The value corresponds to the config ini header `[galaxy_server.{{item}}]` which defines the server details. See *Configuring the ansible-galaxy client* for more details on how to define a Galaxy server. The order of servers in this list is used to as the order in which a collection is resolved. Setting this config option will ignore the *GALAXY_SERVER* config option.

Type
list

Version Added
2.9

Ini

Section
[galaxy]

Key
server_list

Environment

Variable
ANSIBLE_GALAXY_SERVER_LIST

GALAXY_TOKEN_PATH

Description
Local path to galaxy access token file

Type
path

Default
`{{ ANSIBLE_HOME ~ "/galaxy_token" }}`

Version Added
2.9

Ini

Section
[galaxy]

Key
token_path

Environment

Variable
ANSIBLE_GALAXY_TOKEN_PATH

HOST_KEY_CHECKING

Description
Set this to “False” if you want to avoid host key checking by the underlying tools Ansible uses to connect to the host

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
host_key_checking

Environment

Variable
ANSIBLE_HOST_KEY_CHECKING

HOST_PATTERN_MISMATCH

Description
This setting changes the behaviour of mismatched host patterns, it allows you to force a fatal error, a warning or just ignore it

Default
`warning`

Choices

- **error**
  issue a ‘fatal’ error and stop the play
- **warning**
  issue a warning but continue
- **ignore**
  just continue silently

Version Added
2.8

Ini

Section
[inventory]

Key
host_pattern_mismatch

Environment

Variable
ANSIBLE_HOST_PATTERN_MISMATCH

INJECT_FACTS_AS_VARS

Description
Facts are available inside the *ansible_facts* variable, this setting also pushes them as their own vars in the main namespace. Unlike inside the *ansible_facts* dictionary, these will have an *ansible_* prefix.

Type
boolean

Default
`True`

Version Added
2.5

Ini

Section
[defaults]

Key
inject_facts_as_vars

Environment

Variable
ANSIBLE_INJECT_FACT_VARS

INTERPRETER_PYTHON

Description
Path to the Python interpreter to be used for module execution on remote targets, or an automatic discovery mode. Supported discovery modes are `auto` (the default), `auto_silent`, `auto_legacy`, and `auto_legacy_silent`. All discovery modes employ a lookup table to use the included system Python (on distributions known to include one), falling back to a fixed ordered list of well-known Python interpreter locations if a platform-specific default is not available. The fallback behavior will issue a warning that the interpreter should be set explicitly (since interpreters installed later may change which one is used). This warning behavior can be disabled by setting `auto_silent` or `auto_legacy_silent`. The value of `auto_legacy` provides all the same behavior, but for backwards-compatibility with older Ansible releases that always defaulted to `/usr/bin/python`, will use that interpreter if present.

Default
`auto`

Version Added
2.8

Ini

Section
[defaults]

Key
interpreter_python

Environment

Variable
ANSIBLE_PYTHON_INTERPRETER

Variables

name
ansible_python_interpreter

INTERPRETER_PYTHON_FALLBACK

Type
list

Default
`['python3.11', 'python3.10', 'python3.9', 'python3.8', 'python3.7', 'python3.6', 'python3.5', '/usr/bin/python3', '/usr/libexec/platform-python', 'python2.7', '/usr/bin/python', 'python']`

Version Added
2.8

Variables

name
ansible_interpreter_python_fallback

INVALID_TASK_ATTRIBUTE_FAILED

Description
If 'false', invalid attributes for a task will result in warnings instead of errors

Type
boolean

Default
`True`

Version Added
2.7

Ini

Section
[defaults]

Key
invalid_task_attribute_failed

Environment

Variable
ANSIBLE_INVALID_TASK_ATTRIBUTE_FAILED

INVENTORY_ANY_UNPARSED_IS_FAILED

Description
If 'true', it is a fatal error when any given inventory source cannot be successfully parsed by any available inventory plugin; otherwise, this situation only attracts a warning.

Type
boolean

Default
`False`

Version Added
2.7

Ini

Section
[inventory]

Key
any_unparsed_is_failed

Environment

Variable
ANSIBLE_INVENTORY_ANY_UNPARSED_IS_FAILED

INVENTORY_CACHE_ENABLED

Description
Toggle to turn on inventory caching. This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*. The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory configuration. This message will be removed in 2.16.

Type
bool

Default
`False`

Ini

Section
[inventory]

Key
cache

Environment

Variable
ANSIBLE_INVENTORY_CACHE

INVENTORY_CACHE_PLUGIN

Description
The plugin for caching inventory. This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*. The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration. This message will be removed in 2.16.

Ini

Section
[inventory]

Key
cache_plugin

Environment

Variable
ANSIBLE_INVENTORY_CACHE_PLUGIN

INVENTORY_CACHE_PLUGIN_CONNECTION

Description
The inventory cache connection. This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*. The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration. This message will be removed in 2.16.

Ini

Section
[inventory]

Key
cache_connection

Environment

Variable
ANSIBLE_INVENTORY_CACHE_CONNECTION

INVENTORY_CACHE_PLUGIN_PREFIX

Description
The table prefix for the cache plugin. This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*. The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration. This message will be removed in 2.16.

Default
`ansible_inventory_`

Ini

Section
[inventory]

Key
cache_prefix

Environment

Variable
ANSIBLE_INVENTORY_CACHE_PLUGIN_PREFIX

INVENTORY_CACHE_TIMEOUT

Description
Expiration timeout for the inventory cache plugin data. This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*. The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration. This message will be removed in 2.16.

Default
`3600`

Ini

Section
[inventory]

Key
cache_timeout

Environment

Variable
ANSIBLE_INVENTORY_CACHE_TIMEOUT

INVENTORY_ENABLED

Description
List of enabled inventory plugins, it also determines the order in which they are used.

Type
list

Default
`['host_list', 'script', 'auto', 'yaml', 'ini', 'toml']`

Ini

Section
[inventory]

Key
enable_plugins

Environment

Variable
ANSIBLE_INVENTORY_ENABLED

INVENTORY_EXPORT

Description
Controls if ansible-inventory will accurately reflect Ansible's view into inventory or its optimized for exporting.

Type
bool

Default
`False`

Ini

Section
[inventory]

Key
export

Environment

Variable
ANSIBLE_INVENTORY_EXPORT

INVENTORY_IGNORE_EXTS

Description
List of extensions to ignore when using a directory as an inventory source

Type
list

Default
`{{(REJECT_EXTS + ('.orig', '.ini', '.cfg', '.retry'))}}`

Ini

- **Section**
  [defaults]

  Key
  inventory_ignore_extensions

- **Section**
  [inventory]

  Key
  ignore_extensions

Environment

Variable
ANSIBLE_INVENTORY_IGNORE

INVENTORY_IGNORE_PATTERNS

Description
List of patterns to ignore when using a directory as an inventory source

Type
list

Default
`[]`

Ini

- **Section**
  [defaults]

  Key
  inventory_ignore_patterns

- **Section**
  [inventory]

  Key
  ignore_patterns

Environment

Variable
ANSIBLE_INVENTORY_IGNORE_REGEX

INVENTORY_UNPARSED_IS_FAILED

Description
If 'true' it is a fatal error if every single potential inventory source fails to parse, otherwise this situation will only attract a warning.

Type
bool

Default
`False`

Ini

Section
[inventory]

Key
unparsed_is_failed

Environment

Variable
ANSIBLE_INVENTORY_UNPARSED_FAILED

INVENTORY_UNPARSED_WARNING

Description
By default Ansible will issue a warning when no inventory was loaded and notes that it will use an implicit localhost-only inventory. These warnings can be silenced by adjusting this setting to False.

Type
boolean

Default
`True`

Version Added
2.14

Ini

Section
[inventory]

Key
inventory_unparsed_warning

Environment

Variable
ANSIBLE_INVENTORY_UNPARSED_WARNING

JINJA2_NATIVE_WARNING

Description
Toggle to control showing warnings related to running a Jinja version older than required for jinja2_native

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
jinja2_native_warning

Environment

Variable
ANSIBLE_JINJA2_NATIVE_WARNING :Deprecated in: 2.17 :Deprecated detail: This option is no longer used in the Ansible Core code base.

LOCALHOST_WARNING

Description
By default Ansible will issue a warning when there are no hosts in the inventory. These warnings can be silenced by adjusting this setting to False.

Type
boolean

Default
`True`

Version Added
2.6

Ini

Section
[defaults]

Key
localhost_warning

Environment

Variable
ANSIBLE_LOCALHOST_WARNING

MAX_FILE_SIZE_FOR_DIFF

Description
Maximum size of files to be considered for diff display

Type
int

Default
`104448`

Ini

Section
[defaults]

Key
max_diff_size

Environment

Variable
ANSIBLE_MAX_DIFF_SIZE

MODULE_IGNORE_EXTS

Description
List of extensions to ignore when looking for modules to load This is for rejecting script and binary module fallback extensions

Type
list

Default
`{{(REJECT_EXTS + ('.yaml', '.yml', '.ini'))}}`

Ini

Section
[defaults]

Key
module_ignore_exts

Environment

Variable
ANSIBLE_MODULE_IGNORE_EXTS

MODULE_STRICT_UTF8_RESPONSE

Description
Enables whether module responses are evaluated for containing non UTF-8 data Disabling this may result in unexpected behavior Only ansible-core should evaluate this configuration

Type
bool

Default
`True`

Ini

Section
[defaults]

Key
module_strict_utf8_response

Environment

Variable
ANSIBLE_MODULE_STRICT_UTF8_RESPONSE

NETCONF_SSH_CONFIG

Description
This variable is used to enable bastion/jump host with netconf connection. If set to True the bastion/jump host ssh settings should be present in ~/.ssh/config file, alternatively it can be set to custom ssh configuration file path to read the bastion/jump host settings.

Default
`None`

Ini

Section
[netconf_connection]

Key
ssh_config

Environment

Variable
ANSIBLE_NETCONF_SSH_CONFIG

NETWORK_GROUP_MODULES

Type
list

Default
`['eos', 'nxos', 'ios', 'iosxr', 'junos', 'enos', 'ce', 'vyos', 'sros', 'dellos9', 'dellos10', 'dellos6', 'asa', 'aruba', 'aireos', 'bigip', 'ironware', 'onyx', 'netconf', 'exos', 'voss', 'slxos']`

Ini

Section
[defaults]

Key
network_group_modules

Environment

Variable
ANSIBLE_NETWORK_GROUP_MODULES

OLD_PLUGIN_CACHE_CLEARING

Description
Previously Ansible would only clear some of the plugin loading caches when loading new roles, this led to some behaviours in which a plugin loaded in previous plays would be unexpectedly 'sticky'. This setting allows to return to that behaviour.

Type
boolean

Default
`False`

Version Added
2.8

Ini

Section
[defaults]

Key
old_plugin_cache_clear

Environment

Variable
ANSIBLE_OLD_PLUGIN_CACHE_CLEAR

PAGER

Default
`less`

Ini

Section
[defaults]

Key
pager :Version Added: 2.15

Environment

- **Variable**
  ANSIBLE_PAGER

  Version Added
  2.15
- **Variable**
  PAGER

PARAMIKO_HOST_KEY_AUTO_ADD

Type
boolean

Default
`False`

Ini

Section
[paramiko_connection]

Key
host_key_auto_add

Environment

Variable
ANSIBLE_PARAMIKO_HOST_KEY_AUTO_ADD

PARAMIKO_LOOK_FOR_KEYS

Type
boolean

Default
`True`

Ini

Section
[paramiko_connection]

Key
look_for_keys

Environment

Variable
ANSIBLE_PARAMIKO_LOOK_FOR_KEYS

PERSISTENT_COMMAND_TIMEOUT

Description
This controls the amount of time to wait for response from remote device before timing out persistent connection.

Type
int

Default
`30`

Ini

Section
[persistent_connection]

Key
command_timeout

Environment

Variable
ANSIBLE_PERSISTENT_COMMAND_TIMEOUT

PERSISTENT_CONNECT_RETRY_TIMEOUT

Description
This controls the retry timeout for persistent connection to connect to the local domain socket.

Type
integer

Default
`15`

Ini

Section
[persistent_connection]

Key
connect_retry_timeout

Environment

Variable
ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT

PERSISTENT_CONNECT_TIMEOUT

Description
This controls how long the persistent connection will remain idle before it is destroyed.

Type
integer

Default
`30`

Ini

Section
[persistent_connection]

Key
connect_timeout

Environment

Variable
ANSIBLE_PERSISTENT_CONNECT_TIMEOUT

PERSISTENT_CONTROL_PATH_DIR

Description
Path to socket to be used by the connection persistence system.

Type
path

Default
`{{ ANSIBLE_HOME ~ "/pc" }}`

Ini

Section
[persistent_connection]

Key
control_path_dir

Environment

Variable
ANSIBLE_PERSISTENT_CONTROL_PATH_DIR

PLAYBOOK_DIR

Description
A number of non-playbook CLIs have a `--playbook-dir` argument; this sets the default value for it.

Type
path

Version Added
2.9

Ini

Section
[defaults]

Key
playbook_dir

Environment

Variable
ANSIBLE_PLAYBOOK_DIR

PLAYBOOK_VARS_ROOT

Description
This sets which playbook dirs will be used as a root to process vars plugins, which includes finding host_vars/group_vars

Default
`top`

Choices

- **top**
  follows the traditional behavior of using the top playbook in the chain to find the root directory.
- **bottom**
  follows the 2.4.0 behavior of using the current playbook to find the root directory.
- **all**
  examines from the first parent to the current playbook.

Version Added
2.4.1

Ini

Section
[defaults]

Key
playbook_vars_root

Environment

Variable
ANSIBLE_PLAYBOOK_VARS_ROOT

PLUGIN_FILTERS_CFG

Description
A path to configuration for filtering which plugins installed on the system are allowed to be used. See *Rejecting modules* for details of the filter file's format. The default is /etc/ansible/plugin_filters.yml

Type
path

Default
`None`

Version Added
2.5.0

Ini

Section
[defaults]

Key
plugin_filters_cfg

PYTHON_MODULE_RLIMIT_NOFILE

Description
Attempts to set RLIMIT_NOFILE soft limit to the specified value when executing Python modules (can speed up subprocess usage on Python 2.x. See https://bugs.python.org/issue11284). The value will be limited by the existing hard limit. Default value of 0 does not attempt to adjust existing system-defined limits.

Default
`0`

Version Added
2.8

Ini

Section
[defaults]

Key
python_module_rlimit_nofile

Environment

Variable
ANSIBLE_PYTHON_MODULE_RLIMIT_NOFILE

Variables

name
ansible_python_module_rlimit_nofile

RETRY_FILES_ENABLED

Description
This controls whether a failed Ansible playbook should create a .retry file.

Type
bool

Default
`False`

Ini

Section
[defaults]

Key
retry_files_enabled

Environment

Variable
ANSIBLE_RETRY_FILES_ENABLED

RETRY_FILES_SAVE_PATH

Description
This sets the path in which Ansible will save .retry files when a playbook fails and retry files are enabled. This file will be overwritten after each run with the list of failed hosts from all plays.

Type
path

Default
`None`

Ini

Section
[defaults]

Key
retry_files_save_path

Environment

Variable
ANSIBLE_RETRY_FILES_SAVE_PATH

RUN_VARS_PLUGINS

Description
This setting can be used to optimize vars_plugin usage depending on user's inventory size and play selection.

Type
str

Default
`demand`

Choices

- **demand**
  will run vars_plugins relative to inventory sources anytime vars are 'demanded' by tasks.
- **start**
  will run vars_plugins relative to inventory sources after importing that inventory source.

Version Added
2.10

Ini

Section
[defaults]

Key
run_vars_plugins

Environment

Variable
ANSIBLE_RUN_VARS_PLUGINS

SHOW_CUSTOM_STATS

Description
This adds the custom stats set via the set_stats plugin to the default output

Type
bool

Default
`False`

Ini

Section
[defaults]

Key
show_custom_stats

Environment

Variable
ANSIBLE_SHOW_CUSTOM_STATS

STRING_CONVERSION_ACTION

Description
Action to take when a module parameter value is converted to a string (this does not affect variables). For string parameters, values such as '1.00', "['a', 'b',]", and 'yes', 'y', etc. will be converted by the YAML parser unless fully quoted. Valid options are 'error', 'warn', and 'ignore'. Since 2.8, this option defaults to 'warn' but will change to 'error' in 2.12.

Type
string

Default
`warn`

Version Added
2.8

Ini

Section
[defaults]

Key
string_conversion_action

Environment

Variable
ANSIBLE_STRING_CONVERSION_ACTION

STRING_TYPE_FILTERS

Description
This list of filters avoids 'type conversion' when templating variables Useful when you want to avoid conversion into lists or dictionaries for JSON strings, for example.

Type
list

Default
`['string', 'to_json', 'to_nice_json', 'to_yaml', 'to_nice_yaml', 'ppretty', 'json']`

Ini

Section
[jinja2]

Key
dont_type_filters

Environment

Variable
ANSIBLE_STRING_TYPE_FILTERS

SYSTEM_WARNINGS

Description
Allows disabling of warnings related to potential issues on the system running ansible itself (not on the managed hosts) These may include warnings about 3rd party packages or other conditions that should be resolved if possible.

Type
boolean

Default
`True`

Ini

Section
[defaults]

Key
system_warnings

Environment

Variable
ANSIBLE_SYSTEM_WARNINGS

TAGS_RUN

Description
default list of tags to run in your plays, Skip Tags has precedence.

Type
list

Default
`[]`

Version Added
2.5

Ini

Section
[tags]

Key
run

Environment

Variable
ANSIBLE_RUN_TAGS

TAGS_SKIP

Description
default list of tags to skip in your plays, has precedence over Run Tags

Type
list

Default
`[]`

Version Added
2.5

Ini

Section
[tags]

Key
skip

Environment

Variable
ANSIBLE_SKIP_TAGS

TASK_DEBUGGER_IGNORE_ERRORS

Description
This option defines whether the task debugger will be invoked on a failed task when ignore_errors=True is specified. True specifies that the debugger will honor ignore_errors, False will not honor ignore_errors.

Type
boolean

Default
`True`

Version Added
2.7

Ini

Section
[defaults]

Key
task_debugger_ignore_errors

Environment

Variable
ANSIBLE_TASK_DEBUGGER_IGNORE_ERRORS

TASK_TIMEOUT

Description
Set the maximum time (in seconds) that a task can run for. If set to 0 (the default) there is no timeout.

Type
integer

Default
`0`

Version Added
2.10

Ini

Section
[defaults]

Key
task_timeout

Environment

Variable
ANSIBLE_TASK_TIMEOUT

TRANSFORM_INVALID_GROUP_CHARS

Description
Make ansible transform invalid characters in group names supplied by inventory sources.

Type
string

Default
`never`

Choices

- **always**
  it will replace any invalid characters with '_' (underscore) and warn the user
- **never**
  it will allow for the group name but warn about the issue
- **ignore**
  it does the same as 'never', without issuing a warning
- **silently**
  it does the same as 'always', without issuing a warning

Version Added
2.8

Ini

Section
[defaults]

Key
force_valid_group_names

Environment

Variable
ANSIBLE_TRANSFORM_INVALID_GROUP_CHARS

USE_PERSISTENT_CONNECTIONS

Description
Toggles the use of persistence for connections.

Type
boolean

Default
`False`

Ini

Section
[defaults]

Key
use_persistent_connections

Environment

Variable
ANSIBLE_USE_PERSISTENT_CONNECTIONS

VALIDATE_ACTION_GROUP_METADATA

Description
A toggle to disable validating a collection's 'metadata' entry for a module_defaults action group. Metadata containing unexpected fields or value types will produce a warning when this is True.

Type
bool

Default
`True`

Version Added
2.12

Ini

Section
[defaults]

Key
validate_action_group_metadata

Environment

Variable
ANSIBLE_VALIDATE_ACTION_GROUP_METADATA

VARIABLE_PLUGINS_ENABLED

Description
Accept list for variable plugins that require it.

Type
list

Default
`['host_group_vars']`

Version Added
2.10

Ini

Section
[defaults]

Key
vars_plugins_enabled

Environment

Variable
ANSIBLE_VARS_ENABLED

VARIABLE_PRECEDENCE

Description
Allows to change the group variable precedence merge order.

Type
list

Default
`['all_inventory', 'groups_inventory', 'all_plugins_inventory', 'all_plugins_play', 'groups_plugins_inventory', 'groups_plugins_play']`

Version Added
2.4

Ini

Section
[defaults]

Key
precedence

Environment

Variable
ANSIBLE_PRECEDENCE

VAULT_ENCRYPT_SALT

Description
The salt to use for the vault encryption. If it is not provided, a random salt will be used.

Default
`None`

Version Added
2.15

Ini

Section
[defaults]

Key
vault_encrypt_salt

Environment

Variable
ANSIBLE_VAULT_ENCRYPT_SALT

VERBOSE_TO_STDERR

Description
Force 'verbose' option to use stderr instead of stdout

Type
bool

Default
`False`

Version Added
2.8

Ini

Section
[defaults]

Key
verbose_to_stderr

Environment

Variable
ANSIBLE_VERBOSE_TO_STDERR

WIN_ASYNC_STARTUP_TIMEOUT

Description
For asynchronous tasks in Ansible (covered in Asynchronous Actions and Polling), this is how long, in seconds, to wait for the task spawned by Ansible to connect back to the named pipe used on Windows systems. The default is 5 seconds. This can be too low on slower systems, or systems under heavy load. This is not the total time an async command can run for, but is a separate timeout to wait for an async command to start. The task will only start to be timed against its async_timeout once it has connected to the pipe, so the overall maximum duration the task can take will be extended by the amount specified here.

Type
integer

Default
5

Version Added
2.10

Ini

Section
[defaults]

Key
win_async_startup_timeout

Environment

Variable
ANSIBLE_WIN_ASYNC_STARTUP_TIMEOUT

Variables

name
ansible_win_async_startup_timeout

WORKER_SHUTDOWN_POLL_COUNT

Description
The maximum number of times to check Task Queue Manager worker processes to verify they have exited cleanly. After this limit is reached any worker processes still running will be terminated. This is for internal use only.

Type
integer

Default
`0`

Version Added
2.10

Environment

Variable
ANSIBLE_WORKER_SHUTDOWN_POLL_COUNT

WORKER_SHUTDOWN_POLL_DELAY

Description
The number of seconds to sleep between polling loops when checking Task Queue Manager worker processes to verify they have exited cleanly. This is for internal use only.

Type
float

Default
`0.1`

Version Added
2.10

Environment

Variable
ANSIBLE_WORKER_SHUTDOWN_POLL_DELAY

YAML_FILENAME_EXTENSIONS

Description
Check all of these extensions when looking for 'variable' files which should be YAML or JSON or vaulted versions of these. This affects vars_files, include_vars, inventory and vars plugins among others.

Type
list

Default
`['.yml', '.yaml', '.json']`

Ini

Section
[defaults]

Key
yaml_valid_extensions

Environment

Variable
ANSIBLE_YAML_FILENAME_EXT

1.29.3 Environment Variables

Other environment variables to configure plugins in collections can be found in list_of_collection_env_vars.

`ANSIBLE_CONFIG`
Override the default ansible config file

`ANSIBLE_HOME`
The default root path for Ansible config files on the controller.

See also *ANSIBLE_HOME*

`ANSIBLE_CONNECTION_PATH`
Specify where to look for the ansible-connection script. This location will be checked before searching $PATH.If null, ansible will start with the same directory as the ansible script.

See also *ANSIBLE_CONNECTION_PATH*

`ANSIBLE_COW_SELECTION`
This allows you to chose a specific cowsay stencil for the banners or use 'random' to cycle through them.

See also *ANSIBLE_COW_SELECTION*

`ANSIBLE_COW_ACCEPTLIST`
Accept list of cowsay templates that are 'safe' to use, set to empty list if you want to enable all installed templates.

See also *ANSIBLE_COW_ACCEPTLIST*
Version Added
2.11

`ANSIBLE_FORCE_COLOR`
This option forces color mode even when running without a TTY or the "nocolor" setting is True.

See also *ANSIBLE_FORCE_COLOR*

`ANSIBLE_NOCOLOR`
This setting allows suppressing colorizing output, which is used to give a better indication of failure and status information.

See also *ANSIBLE_NOCOLOR*

`NO_COLOR`
This setting allows suppressing colorizing output, which is used to give a better indication of failure and status information.

See also *ANSIBLE_NOCOLOR*
Version Added
2.11

ANSIBLE_NOCOWS

If you have cowsay installed but want to avoid the 'cows' (why????), use this.

See also *ANSIBLE_NOCOWS*

ANSIBLE_COW_PATH

Specify a custom cowsay path or swap in your cowsay implementation of choice

See also *ANSIBLE_COW_PATH*

ANSIBLE_PIPELINING

This is a global option, each connection plugin can override either by having more specific options or not supporting pipelining at all.Pipelining, if supported by the connection plugin, reduces the number of network operations required to execute a module on the remote server, by executing many Ansible modules without actual file transfer.It can result in a very significant performance improvement when enabled.However this conflicts with privilege escalation (become). For example, when using 'sudo:' operations you must first disable 'requiretty' in /etc/sudoers on all managed hosts, which is why it is disabled by default.This setting will be disabled if `ANSIBLE_KEEP_REMOTE_FILES` is enabled.

See also *ANSIBLE_PIPELINING*

ANSIBLE_ANY_ERRORS_FATAL

Sets the default value for the any_errors_fatal keyword, if True, Task failures will be considered fatal errors.

See also *ANY_ERRORS_FATAL*

ANSIBLE_BECOME_ALLOW_SAME_USER

This setting controls if become is skipped when remote user and become user are the same. I.E root sudo to root.If executable, it will be run and the resulting stdout will be used as the password.

See also *BECOME_ALLOW_SAME_USER*

ANSIBLE_BECOME_PASSWORD_FILE

The password file to use for the become plugin. –become-password-file.If executable, it will be run and the resulting stdout will be used as the password.

See also *BECOME_PASSWORD_FILE*

ANSIBLE_AGNOSTIC_BECOME_PROMPT

Display an agnostic become prompt instead of displaying a prompt containing the command line supplied become method

See also *AGNOSTIC_BECOME_PROMPT*

ANSIBLE_CACHE_PLUGIN

Chooses which cache plugin to use, the default 'memory' is ephemeral.

See also *CACHE_PLUGIN*

ANSIBLE_CACHE_PLUGIN_CONNECTION

Defines connection or path information for the cache plugin

See also *CACHE_PLUGIN_CONNECTION*

ANSIBLE_CACHE_PLUGIN_PREFIX

Prefix to use for cache plugin files/tables

See also *CACHE_PLUGIN_PREFIX*

ANSIBLE_CACHE_PLUGIN_TIMEOUT

Expiration timeout for the cache plugin data

See also *CACHE_PLUGIN_TIMEOUT*

ANSIBLE_COLLECTIONS_SCAN_SYS_PATH

A boolean to enable or disable scanning the sys.path for installed collections

See also *COLLECTIONS_SCAN_SYS_PATH*

ANSIBLE_COLLECTIONS_PATHS

Colon separated paths in which Ansible will search for collections content. Collections must be in nested *subdirectories*, not directly in these directories. For example, if `COLLECTIONS_PATHS` includes `'{{ ANSIBLE_HOME ~ "/collections" }}'`, and you want to add `my.collection` to that directory, it must be saved as `'{{ ANSIBLE_HOME} ~ "/collections/ansible_collections/my/collection" }}'`.

See also *COLLECTIONS_PATHS*

ANSIBLE_COLLECTIONS_PATH

Colon separated paths in which Ansible will search for collections content. Collections must be in nested *subdirectories*, not directly in these directories. For example, if `COLLECTIONS_PATHS` includes `'{{ ANSIBLE_HOME ~ "/collections" }}'`, and you want to add `my.collection` to that directory, it must be saved as `'{{ ANSIBLE_HOME} ~ "/collections/ansible_collections/my/collection" }}'`.

See also *COLLECTIONS_PATHS*

Version Added
2.10

ANSIBLE_COLLECTIONS_ON_ANSIBLE_VERSION_MISMATCH

When a collection is loaded that does not support the running Ansible version (with the collection metadata key *requires_ansible*).

See also *COLLECTIONS_ON_ANSIBLE_VERSION_MISMATCH*

ANSIBLE_COLOR_CHANGED

Defines the color to use on 'Changed' task status

See also *COLOR_CHANGED*

ANSIBLE_COLOR_CONSOLE_PROMPT

Defines the default color to use for ansible-console

See also *COLOR_CONSOLE_PROMPT*

ANSIBLE_COLOR_DEBUG

Defines the color to use when emitting debug messages

See also *COLOR_DEBUG*

ANSIBLE_COLOR_DEPRECATE

Defines the color to use when emitting deprecation messages

See also *COLOR_DEPRECATE*

ANSIBLE_COLOR_DIFF_ADD

Defines the color to use when showing added lines in diffs

See also *COLOR_DIFF_ADD*

ANSIBLE_COLOR_DIFF_LINES

Defines the color to use when showing diffs

See also *COLOR_DIFF_LINES*

ANSIBLE_COLOR_DIFF_REMOVE

Defines the color to use when showing removed lines in diffs

See also *COLOR_DIFF_REMOVE*

ANSIBLE_COLOR_ERROR

Defines the color to use when emitting error messages

See also *COLOR_ERROR*

ANSIBLE_COLOR_HIGHLIGHT

Defines the color to use for highlighting

See also *COLOR_HIGHLIGHT*

ANSIBLE_COLOR_OK

Defines the color to use when showing 'OK' task status

See also *COLOR_OK*

ANSIBLE_COLOR_SKIP

Defines the color to use when showing 'Skipped' task status

See also *COLOR_SKIP*

ANSIBLE_COLOR_UNREACHABLE

Defines the color to use on 'Unreachable' status

See also *COLOR_UNREACHABLE*

ANSIBLE_COLOR_VERBOSE

Defines the color to use when emitting verbose messages. i.e those that show with '-v's.

See also *COLOR_VERBOSE*

ANSIBLE_COLOR_WARN

Defines the color to use when emitting warning messages

See also *COLOR_WARN*

ANSIBLE_CONNECTION_PASSWORD_FILE

The password file to use for the connection plugin. –connection-password-file.

See also *CONNECTION_PASSWORD_FILE*

_ANSIBLE_COVERAGE_REMOTE_OUTPUT

Sets the output directory on the remote host to generate coverage reports to.Currently only used for remote coverage on PowerShell modules.This is for internal use only.

See also *COVERAGE_REMOTE_OUTPUT*

_ANSIBLE_COVERAGE_REMOTE_PATH_FILTER

A list of paths for files on the Ansible controller to run coverage for when executing on the remote host.Only files that match the path glob will have its coverage collected.Multiple path globs can be specified and are separated by `:`.Currently only used for remote coverage on PowerShell modules.This is for internal use only.

See also *COVERAGE_REMOTE_PATHS*

ANSIBLE_ACTION_WARNINGS

By default Ansible will issue a warning when received from a task action (module or action plugin)These warnings can be silenced by adjusting this setting to False.

See also *ACTION_WARNINGS*

ANSIBLE_LOCALHOST_WARNING

By default Ansible will issue a warning when there are no hosts in the inventory.These warnings can be silenced by adjusting this setting to False.

See also *LOCALHOST_WARNING*

ANSIBLE_INVENTORY_UNPARSED_WARNING

By default Ansible will issue a warning when no inventory was loaded and notes that it will use an implicit localhost-only inventory.These warnings can be silenced by adjusting this setting to False.

See also *INVENTORY_UNPARSED_WARNING*

ANSIBLE_DOC_FRAGMENT_PLUGINS

Colon separated paths in which Ansible will search for Documentation Fragments Plugins.

See also *DOC_FRAGMENT_PLUGIN_PATH*

ANSIBLE_ACTION_PLUGINS

Colon separated paths in which Ansible will search for Action Plugins.

See also *DEFAULT_ACTION_PLUGIN_PATH*

ANSIBLE_ASK_PASS

This controls whether an Ansible playbook should prompt for a login password. If using SSH keys for authentication, you probably do not need to change this setting.

See also *DEFAULT_ASK_PASS*

ANSIBLE_ASK_VAULT_PASS

This controls whether an Ansible playbook should prompt for a vault password.

See also *DEFAULT_ASK_VAULT_PASS*

ANSIBLE_BECOME

Toggles the use of privilege escalation, allowing you to 'become' another user after login.

See also *DEFAULT_BECOME*

ANSIBLE_BECOME_ASK_PASS

Toggle to prompt for privilege escalation password.

See also *DEFAULT_BECOME_ASK_PASS*

ANSIBLE_BECOME_METHOD

Privilege escalation method to use when *become* is enabled.

See also *DEFAULT_BECOME_METHOD*

ANSIBLE_BECOME_EXE

executable to use for privilege escalation, otherwise Ansible will depend on PATH

See also *DEFAULT_BECOME_EXE*

ANSIBLE_BECOME_FLAGS

Flags to pass to the privilege escalation executable.

See also *DEFAULT_BECOME_FLAGS*

ANSIBLE_BECOME_PLUGINS

Colon separated paths in which Ansible will search for Become Plugins.

See also *BECOME_PLUGIN_PATH*

ANSIBLE_BECOME_USER

The user your login/remote user 'becomes' when using privilege escalation, most systems will use 'root' when no user is specified.

See also *DEFAULT_BECOME_USER*

ANSIBLE_CACHE_PLUGINS

Colon separated paths in which Ansible will search for Cache Plugins.

See also *DEFAULT_CACHE_PLUGIN_PATH*

ANSIBLE_CALLBACK_PLUGINS

Colon separated paths in which Ansible will search for Callback Plugins.

See also *DEFAULT_CALLBACK_PLUGIN_PATH*

ANSIBLE_CALLBACKS_ENABLED

List of enabled callbacks, not all callbacks need enabling, but many of those shipped with Ansible do as we don't want them activated by default.

See also *CALLBACKS_ENABLED*

Version Added
2.11

ANSIBLE_CLICONF_PLUGINS

Colon separated paths in which Ansible will search for Cliconf Plugins.

See also *DEFAULT_CLICONF_PLUGIN_PATH*

ANSIBLE_CONNECTION_PLUGINS

Colon separated paths in which Ansible will search for Connection Plugins.

See also *DEFAULT_CONNECTION_PLUGIN_PATH*

ANSIBLE_DEBUG

Toggles debug output in Ansible. This is *very* verbose and can hinder multiprocessing. Debug output can also include secret information despite no_log settings being enabled, which means debug mode should not be used in production.

See also *DEFAULT_DEBUG*

ANSIBLE_EXECUTABLE

This indicates the command to use to spawn a shell under for Ansible's execution needs on a target. Users may need to change this in rare instances when shell usage is constrained, but in most cases it may be left as is.

See also *DEFAULT_EXECUTABLE*

ANSIBLE_FACT_PATH

This option allows you to globally configure a custom path for 'local_facts' for the implied ansible_collections.ansible.builtin.setup_module task when using fact gathering.If not set, it will fallback to the default from the `ansible.builtin.setup` module: `/etc/ansible/facts.d`.This does **not** affect user defined tasks that use the `ansible.builtin.setup` module.The real action being created by the implicit task is currently `ansible.legacy.gather_facts` module, which then calls the configured fact modules, by default this will be `ansible.builtin.setup` for POSIX systems but other platforms might have different defaults.

See also *DEFAULT_FACT_PATH*

ANSIBLE_FILTER_PLUGINS

Colon separated paths in which Ansible will search for Jinja2 Filter Plugins.

See also *DEFAULT_FILTER_PLUGIN_PATH*

ANSIBLE_FORCE_HANDLERS

This option controls if notified handlers run on a host even if a failure occurs on that host.When false, the handlers will not run if a failure has occurred on a host.This can also be set per play or on the command line. See Handlers and Failure for more details.

See also *DEFAULT_FORCE_HANDLERS*

ANSIBLE_FORKS

Maximum number of forks Ansible will use to execute tasks on target hosts.

See also *DEFAULT_FORKS*

ANSIBLE_GATHERING

This setting controls the default policy of fact gathering (facts discovered about remote systems).This option can be useful for those wishing to save fact gathering time. Both 'smart' and 'explicit' will use the cache plugin.

See also *DEFAULT_GATHERING*

ANSIBLE_GATHER_SUBSET

Set the *gather_subset* option for the ansible_collections.ansible.builtin.setup_module task in the implicit fact gathering. See the module documentation for specifics.It does **not** apply to user defined `ansible.builtin.setup` tasks.

See also *DEFAULT_GATHER_SUBSET*

ANSIBLE_GATHER_TIMEOUT

Set the timeout in seconds for the implicit fact gathering, see the module documentation for specifics.It does **not** apply to user defined ansible_collections.ansible.builtin.setup_module tasks.

See also *DEFAULT_GATHER_TIMEOUT*

ANSIBLE_HASH_BEHAVIOUR

This setting controls how duplicate definitions of dictionary variables (aka hash, map, associative array) are handled in Ansible.This does not affect variables whose values are scalars (integers, strings) or arrays.**WARNING**, changing this setting is not recommended as this is fragile and makes your content (plays, roles, collections) non portable, leading to continual confusion and misuse. Don't change this setting unless you think you have an absolute need for it.We recommend avoiding reusing variable names and relying on the `combine` filter and `vars` and `varnames` lookups to create merged versions of the individual variables. In our experience this is rarely really needed and a sign that too much complexity has been introduced into the data structures and plays.For some uses you can also look into custom vars_plugins to merge on input, even substituting the default `host_group_vars` that is in charge of parsing the `host_vars/` and `group_vars/` directories. Most users of this setting are only interested in inventory scope, but the setting itself affects all sources and makes debugging even harder.All playbooks and roles in the official examples repos assume the default for this setting.Changing the setting to `merge` applies across variable sources, but many sources will internally still overwrite the variables. For example `include_vars` will dedupe variables internally before updating Ansible, with 'last defined' overwriting previous definitions in same file.The Ansible project recommends you **avoid ``merge`` for new projects.**It is the intention of the Ansible developers to eventually deprecate and remove this setting, but it is being kept as some users do heavily rely on it. New projects should **avoid 'merge'**.

See also *DEFAULT_HASH_BEHAVIOUR*

ANSIBLE_INVENTORY

Comma separated list of Ansible inventory sources

See also *DEFAULT_HOST_LIST*

`ANSIBLE_HTTPAPI_PLUGINS`
: Colon separated paths in which Ansible will search for HttpApi Plugins.

  See also *DEFAULT_HTTPAPI_PLUGIN_PATH*

`ANSIBLE_INVENTORY_PLUGINS`
: Colon separated paths in which Ansible will search for Inventory Plugins.

  See also *DEFAULT_INVENTORY_PLUGIN_PATH*

`ANSIBLE_JINJA2_EXTENSIONS`
: This is a developer-specific feature that allows enabling additional Jinja2 extensions.See the Jinja2 documentation for details. If you do not know what these do, you probably don't need to change this setting :)

  See also *DEFAULT_JINJA2_EXTENSIONS*

`ANSIBLE_JINJA2_NATIVE`
: This option preserves variable types during template operations.

  See also *DEFAULT_JINJA2_NATIVE*

`ANSIBLE_KEEP_REMOTE_FILES`
: Enables/disables the cleaning up of the temporary files Ansible used to execute the tasks on the remote.If this option is enabled it will disable `ANSIBLE_PIPELINING`.

  See also *DEFAULT_KEEP_REMOTE_FILES*

`ANSIBLE_LIBVIRT_LXC_NOSECLABEL`
: This setting causes libvirt to connect to lxc containers by passing –noseclabel to virsh. This is necessary when running on systems which do not have SELinux.

  See also *DEFAULT_LIBVIRT_LXC_NOSECLABEL*

`ANSIBLE_LOAD_CALLBACK_PLUGINS`
: Controls whether callback plugins are loaded when running /usr/bin/ansible. This may be used to log activity from the command line, send notifications, and so on. Callback plugins are always loaded for `ansible-playbook`.

  See also *DEFAULT_LOAD_CALLBACK_PLUGINS*

`ANSIBLE_LOCAL_TEMP`
: Temporary directory for Ansible to use on the controller.

  See also *DEFAULT_LOCAL_TMP*

`ANSIBLE_LOG_PATH`
: File to which Ansible will log on the controller. When empty logging is disabled.

  See also *DEFAULT_LOG_PATH*

`ANSIBLE_LOG_FILTER`
: List of logger names to filter out of the log file

  See also *DEFAULT_LOG_FILTER*

`ANSIBLE_LOOKUP_PLUGINS`
: Colon separated paths in which Ansible will search for Lookup Plugins.

  See also *DEFAULT_LOOKUP_PLUGIN_PATH*

ANSIBLE_MODULE_ARGS

This sets the default arguments to pass to the `ansible` adhoc binary if no `-a` is specified.

See also *DEFAULT_MODULE_ARGS*

ANSIBLE_LIBRARY

Colon separated paths in which Ansible will search for Modules.

See also *DEFAULT_MODULE_PATH*

ANSIBLE_MODULE_UTILS

Colon separated paths in which Ansible will search for Module utils files, which are shared by modules.

See also *DEFAULT_MODULE_UTILS_PATH*

ANSIBLE_NETCONF_PLUGINS

Colon separated paths in which Ansible will search for Netconf Plugins.

See also *DEFAULT_NETCONF_PLUGIN_PATH*

ANSIBLE_NO_LOG

Toggle Ansible's display and logging of task details, mainly used to avoid security disclosures.

See also *DEFAULT_NO_LOG*

ANSIBLE_NO_TARGET_SYSLOG

Toggle Ansible logging to syslog on the target when it executes tasks. On Windows hosts this will disable a newer style PowerShell modules from writing to the event log.

See also *DEFAULT_NO_TARGET_SYSLOG*

ANSIBLE_NULL_REPRESENTATION

What templating should return as a 'null' value. When not set it will let Jinja2 decide.

See also *DEFAULT_NULL_REPRESENTATION*

ANSIBLE_POLL_INTERVAL

For asynchronous tasks in Ansible (covered in Asynchronous Actions and Polling), this is how often to check back on the status of those tasks when an explicit poll interval is not supplied. The default is a reasonably moderate 15 seconds which is a tradeoff between checking in frequently and providing a quick turnaround when something may have completed.

See also *DEFAULT_POLL_INTERVAL*

ANSIBLE_PRIVATE_KEY_FILE

Option for connections using a certificate or key file to authenticate, rather than an agent or passwords, you can set the default value here to avoid re-specifying –private-key with every invocation.

See also *DEFAULT_PRIVATE_KEY_FILE*

ANSIBLE_PRIVATE_ROLE_VARS

Makes role variables inaccessible from other roles.This was introduced as a way to reset role variables to default values if a role is used more than once in a playbook.

See also *DEFAULT_PRIVATE_ROLE_VARS*

ANSIBLE_REMOTE_PORT

Port to use in remote connections, when blank it will use the connection plugin default.

See also *DEFAULT_REMOTE_PORT*

ANSIBLE_REMOTE_USER

Sets the login user for the target machinesWhen blank it uses the connection plugin's default, normally the user currently executing Ansible.

See also *DEFAULT_REMOTE_USER*

ANSIBLE_ROLES_PATH

Colon separated paths in which Ansible will search for Roles.

See also *DEFAULT_ROLES_PATH*

ANSIBLE_SELINUX_SPECIAL_FS

Some filesystems do not support safe operations and/or return inconsistent errors, this setting makes Ansible 'tolerate' those in the list w/o causing fatal errors.Data corruption may occur and writes are not always verified when a filesystem is in the list.

See also *DEFAULT_SELINUX_SPECIAL_FS*

Version Added
2.9

ANSIBLE_STDOUT_CALLBACK

Set the main callback used to display Ansible output. You can only have one at a time.You can have many other callbacks, but just one can be in charge of stdout.See *Callback plugins* for a list of available options.

See also *DEFAULT_STDOUT_CALLBACK*

ANSIBLE_EDITOR

See also *EDITOR*

Version Added
2.15

EDITOR

See also *EDITOR*

ANSIBLE_ENABLE_TASK_DEBUGGER

Whether or not to enable the task debugger, this previously was done as a strategy plugin.Now all strategy plugins can inherit this behavior. The debugger defaults to activating whena task is failed on unreachable. Use the debugger keyword for more flexibility.

See also *ENABLE_TASK_DEBUGGER*

ANSIBLE_TASK_DEBUGGER_IGNORE_ERRORS

This option defines whether the task debugger will be invoked on a failed task when ignore_errors=True is specified.True specifies that the debugger will honor ignore_errors, False will not honor ignore_errors.

See also *TASK_DEBUGGER_IGNORE_ERRORS*

ANSIBLE_STRATEGY

Set the default strategy used for plays.

See also *DEFAULT_STRATEGY*

ANSIBLE_STRATEGY_PLUGINS

Colon separated paths in which Ansible will search for Strategy Plugins.

See also *DEFAULT_STRATEGY_PLUGIN_PATH*

ANSIBLE_SU

Toggle the use of "su" for tasks.

See also *DEFAULT_SU*

ANSIBLE_SYSLOG_FACILITY

Syslog facility to use when Ansible logs to the remote target

See also *DEFAULT_SYSLOG_FACILITY*

ANSIBLE_TERMINAL_PLUGINS

Colon separated paths in which Ansible will search for Terminal Plugins.

See also *DEFAULT_TERMINAL_PLUGIN_PATH*

ANSIBLE_TEST_PLUGINS

Colon separated paths in which Ansible will search for Jinja2 Test Plugins.

See also *DEFAULT_TEST_PLUGIN_PATH*

ANSIBLE_TIMEOUT

This is the default timeout for connection plugins to use.

See also *DEFAULT_TIMEOUT*

ANSIBLE_TRANSPORT

Default connection plugin to use, the 'smart' option will toggle between 'ssh' and 'paramiko' depending on controller OS and ssh versions

See also *DEFAULT_TRANSPORT*

ANSIBLE_ERROR_ON_UNDEFINED_VARS

When True, this causes ansible templating to fail steps that reference variable names that are likely typoed.Otherwise, any '{{ template_expression }}' that contains undefined variables will be rendered in a template or ansible action line exactly as written.

See also *DEFAULT_UNDEFINED_VAR_BEHAVIOR*

ANSIBLE_VARS_PLUGINS

Colon separated paths in which Ansible will search for Vars Plugins.

See also *DEFAULT_VARS_PLUGIN_PATH*

ANSIBLE_VAULT_ID_MATCH

If true, decrypting vaults with a vault id will only try the password from the matching vault-id

See also *DEFAULT_VAULT_ID_MATCH*

ANSIBLE_VAULT_IDENTITY

The label to use for the default vault id label in cases where a vault id label is not provided

See also *DEFAULT_VAULT_IDENTITY*

ANSIBLE_VAULT_ENCRYPT_SALT

The salt to use for the vault encryption. If it is not provided, a random salt will be used.

See also *VAULT_ENCRYPT_SALT*

ANSIBLE_VAULT_ENCRYPT_IDENTITY

The vault_id to use for encrypting by default. If multiple vault_ids are provided, this specifies which to use for encryption. The –encrypt-vault-id cli option overrides the configured value.

See also *DEFAULT_VAULT_ENCRYPT_IDENTITY*

ANSIBLE_VAULT_IDENTITY_LIST

A list of vault-ids to use by default. Equivalent to multiple –vault-id args. Vault-ids are tried in order.

See also *DEFAULT_VAULT_IDENTITY_LIST*

ANSIBLE_VAULT_PASSWORD_FILE

The vault password file to use. Equivalent to –vault-password-file or –vault-idIf executable, it will be run and the resulting stdout will be used as the password.

See also *DEFAULT_VAULT_PASSWORD_FILE*

ANSIBLE_VERBOSITY

Sets the default verbosity, equivalent to the number of `-v` passed in the command line.

See also *DEFAULT_VERBOSITY*

ANSIBLE_DEPRECATION_WARNINGS

Toggle to control the showing of deprecation warnings

See also *DEPRECATION_WARNINGS*

ANSIBLE_DEVEL_WARNING

Toggle to control showing warnings related to running devel

See also *DEVEL_WARNING*

ANSIBLE_DIFF_ALWAYS

Configuration toggle to tell modules to show differences when in 'changed' status, equivalent to `--diff`.

See also *DIFF_ALWAYS*

ANSIBLE_DIFF_CONTEXT

How many lines of context to show when displaying the differences between files.

See also *DIFF_CONTEXT*

ANSIBLE_DISPLAY_ARGS_TO_STDOUT

Normally `ansible-playbook` will print a header for each task that is run. These headers will contain the name: field from the task if you specified one. If you didn't then `ansible-playbook` uses the task's action to help you tell which task is presently running. Sometimes you run many of the same action and so you want more information about the task to differentiate it from others of the same action. If you set this variable to True in the config then `ansible-playbook` will also include the task's arguments in the header.This setting defaults to False because there is a chance that you have sensitive values in your parameters and you do not want those to be printed.If you set this to True you should be sure that you have secured your environment's stdout (no one can shoulder surf your screen and you aren't saving stdout to an insecure file) or made sure that all of your playbooks explicitly added the `no_log: True` parameter to tasks which have sensitive values See How do I keep secret data in my playbook? for more information.

See also *DISPLAY_ARGS_TO_STDOUT*

ANSIBLE_DISPLAY_SKIPPED_HOSTS

Toggle to control displaying skipped task/host entries in a task in the default callback

See also *DISPLAY_SKIPPED_HOSTS*

ANSIBLE_DUPLICATE_YAML_DICT_KEY

By default Ansible will issue a warning when a duplicate dict key is encountered in YAML.These warnings can be silenced by adjusting this setting to False.

See also *DUPLICATE_YAML_DICT_KEY*

ANSIBLE_ERROR_ON_MISSING_HANDLER

Toggle to allow missing handlers to become a warning instead of an error when notifying.

See also *ERROR_ON_MISSING_HANDLER*

ANSIBLE_FACTS_MODULES

Which modules to run during a play's fact gathering stage, using the default of 'smart' will try to figure it out based on connection type.If adding your own modules but you still want to use the default Ansible facts, you will want to include 'setup' or corresponding network module to the list (if you add 'smart', Ansible will also figure it out).This does not affect explicit calls to the 'setup' module, but does always affect the 'gather_facts' action (implicit or explicit).

See also *FACTS_MODULES*

ANSIBLE_GALAXY_IGNORE

If set to yes, ansible-galaxy will not validate TLS certificates. This can be useful for testing against a server with a self-signed certificate.

See also *GALAXY_IGNORE_CERTS*

ANSIBLE_GALAXY_ROLE_SKELETON

Role skeleton directory to use as a template for the `init` action in `ansible-galaxy`/`ansible-galaxy role`, same as `--role-skeleton`.

See also *GALAXY_ROLE_SKELETON*

ANSIBLE_GALAXY_ROLE_SKELETON_IGNORE

patterns of files to ignore inside a Galaxy role or collection skeleton directory

See also *GALAXY_ROLE_SKELETON_IGNORE*

ANSIBLE_GALAXY_COLLECTION_SKELETON

Collection skeleton directory to use as a template for the `init` action in `ansible-galaxy collection`, same as `--collection-skeleton`.

See also *GALAXY_COLLECTION_SKELETON*

ANSIBLE_GALAXY_COLLECTION_SKELETON_IGNORE

patterns of files to ignore inside a Galaxy collection skeleton directory

See also *GALAXY_COLLECTION_SKELETON_IGNORE*

ANSIBLE_GALAXY_SERVER

URL to prepend when roles don't specify the full URI, assume they are referencing this server as the source.

See also *GALAXY_SERVER*

ANSIBLE_GALAXY_SERVER_LIST

A list of Galaxy servers to use when installing a collection.The value corresponds to the config ini header `[galaxy_server.{{item}}]` which defines the server details.See *Configuring the ansible-galaxy client* for more details on how to define a Galaxy server.The order of servers in this list is used to as the order in which a collection is resolved.Setting this config option will ignore the *GALAXY_SERVER* config option.

See also *GALAXY_SERVER_LIST*

ANSIBLE_GALAXY_TOKEN_PATH

Local path to galaxy access token file

See also *GALAXY_TOKEN_PATH*

ANSIBLE_GALAXY_DISPLAY_PROGRESS

Some steps in `ansible-galaxy` display a progress wheel which can cause issues on certain displays or when outputting the stdout to a file.This config option controls whether the display wheel is shown or not.The default is to show the display wheel if stdout has a tty.

See also *GALAXY_DISPLAY_PROGRESS*

ANSIBLE_GALAXY_CACHE_DIR

The directory that stores cached responses from a Galaxy server.This is only used by the `ansible-galaxy collection install` and `download` commands.Cache files inside this dir will be ignored if they are world writable.

See also *GALAXY_CACHE_DIR*

ANSIBLE_GALAXY_DISABLE_GPG_VERIFY

Disable GPG signature verification during collection installation.

See also *GALAXY_DISABLE_GPG_VERIFY*

ANSIBLE_GALAXY_GPG_KEYRING

Configure the keyring used for GPG signature verification during collection installation and verification.

See also *GALAXY_GPG_KEYRING*

ANSIBLE_GALAXY_IGNORE_SIGNATURE_STATUS_CODES

A list of GPG status codes to ignore during GPG signature verification. See L(https://github.com/gpg/gnupg/blob/master/doc/DETAILS#general-status-codes) for status code descriptions.If fewer signatures successfully verify the collection than *GALAXY_REQUIRED_VALID_SIGNATURE_COUNT*, signature verification will fail even if all error codes are ignored.

See also *GALAXY_IGNORE_INVALID_SIGNATURE_STATUS_CODES*

ANSIBLE_GALAXY_REQUIRED_VALID_SIGNATURE_COUNT

The number of signatures that must be successful during GPG signature verification while installing or verifying collections.This should be a positive integer or all to indicate all signatures must successfully validate the collection.Prepend + to the value to fail if no valid signatures are found for the collection.

See also *GALAXY_REQUIRED_VALID_SIGNATURE_COUNT*

ANSIBLE_HOST_KEY_CHECKING

Set this to "False" if you want to avoid host key checking by the underlying tools Ansible uses to connect to the host

See also *HOST_KEY_CHECKING*

ANSIBLE_HOST_PATTERN_MISMATCH

This setting changes the behaviour of mismatched host patterns, it allows you to force a fatal error, a warning or just ignore it

See also *HOST_PATTERN_MISMATCH*

ANSIBLE_PYTHON_INTERPRETER

Path to the Python interpreter to be used for module execution on remote targets, or an automatic discovery mode. Supported discovery modes are `auto` (the default), `auto_silent`, `auto_legacy`, and `auto_legacy_silent`. All discovery modes employ a lookup table to use the included system Python (on distributions known to include one), falling back to a fixed ordered list of well-known Python interpreter locations if a platform-specific default is not available. The fallback behavior will issue a warning that the interpreter should be set explicitly (since interpreters installed later may change which one is used). This warning behavior can be disabled by setting `auto_silent` or `auto_legacy_silent`. The value of `auto_legacy` provides all the same behavior, but for backwards-compatibility with older Ansible releases that always defaulted to `/usr/bin/python`, will use that interpreter if present.

See also *INTERPRETER_PYTHON*

ANSIBLE_TRANSFORM_INVALID_GROUP_CHARS

Make ansible transform invalid characters in group names supplied by inventory sources.

See also *TRANSFORM_INVALID_GROUP_CHARS*

ANSIBLE_INVALID_TASK_ATTRIBUTE_FAILED

If 'false', invalid attributes for a task will result in warnings instead of errors

See also *INVALID_TASK_ATTRIBUTE_FAILED*

ANSIBLE_INVENTORY_ANY_UNPARSED_IS_FAILED

If 'true', it is a fatal error when any given inventory source cannot be successfully parsed by any available inventory plugin; otherwise, this situation only attracts a warning.

See also *INVENTORY_ANY_UNPARSED_IS_FAILED*

ANSIBLE_INVENTORY_CACHE

Toggle to turn on inventory caching.This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*.The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory configuration.This message will be removed in 2.16.

See also *INVENTORY_CACHE_ENABLED*

ANSIBLE_INVENTORY_CACHE_PLUGIN

The plugin for caching inventory.This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*.The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration.This message will be removed in 2.16.

See also *INVENTORY_CACHE_PLUGIN*

ANSIBLE_INVENTORY_CACHE_CONNECTION

The inventory cache connection.This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*.The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration.This message will be removed in 2.16.

See also *INVENTORY_CACHE_PLUGIN_CONNECTION*

ANSIBLE_INVENTORY_CACHE_PLUGIN_PREFIX

The table prefix for the cache plugin.This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*.The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration.This message will be removed in 2.16.

See also *INVENTORY_CACHE_PLUGIN_PREFIX*

ANSIBLE_INVENTORY_CACHE_TIMEOUT

Expiration timeout for the inventory cache plugin data.This setting has been moved to the individual inventory plugins as a plugin option *Inventory plugins*.The existing configuration settings are still accepted with the inventory plugin adding additional options from inventory and fact cache configuration.This message will be removed in 2.16.

See also *INVENTORY_CACHE_TIMEOUT*

ANSIBLE_INVENTORY_ENABLED

List of enabled inventory plugins, it also determines the order in which they are used.

See also *INVENTORY_ENABLED*

ANSIBLE_INVENTORY_EXPORT

Controls if ansible-inventory will accurately reflect Ansible's view into inventory or its optimized for exporting.

See also *INVENTORY_EXPORT*

ANSIBLE_INVENTORY_IGNORE

List of extensions to ignore when using a directory as an inventory source

See also *INVENTORY_IGNORE_EXTS*

ANSIBLE_INVENTORY_IGNORE_REGEX

List of patterns to ignore when using a directory as an inventory source

See also *INVENTORY_IGNORE_PATTERNS*

ANSIBLE_INVENTORY_UNPARSED_FAILED

If 'true' it is a fatal error if every single potential inventory source fails to parse, otherwise this situation will only attract a warning.

See also *INVENTORY_UNPARSED_IS_FAILED*

ANSIBLE_JINJA2_NATIVE_WARNING

Toggle to control showing warnings related to running a Jinja version older than required for jinja2_native

See also *JINJA2_NATIVE_WARNING*

Deprecated in
2.17

Deprecated detail
This option is no longer used in the Ansible Core code base.

ANSIBLE_MAX_DIFF_SIZE

Maximum size of files to be considered for diff display

See also *MAX_FILE_SIZE_FOR_DIFF*

ANSIBLE_NETWORK_GROUP_MODULES

See also *NETWORK_GROUP_MODULES*

ANSIBLE_INJECT_FACT_VARS

Facts are available inside the *ansible_facts* variable, this setting also pushes them as their own vars in the main namespace.Unlike inside the *ansible_facts* dictionary, these will have an *ansible_* prefix.

See also *INJECT_FACTS_AS_VARS*

ANSIBLE_MODULE_IGNORE_EXTS

List of extensions to ignore when looking for modules to loadThis is for rejecting script and binary module fallback extensions

See also *MODULE_IGNORE_EXTS*

ANSIBLE_MODULE_STRICT_UTF8_RESPONSE

Enables whether module responses are evaluated for containing non UTF-8 dataDisabling this may result in unexpected behaviorOnly ansible-core should evaluate this configuration

See also *MODULE_STRICT_UTF8_RESPONSE*

ANSIBLE_OLD_PLUGIN_CACHE_CLEAR

Previously Ansible would only clear some of the plugin loading caches when loading new roles, this led to some behaviours in which a plugin loaded in previous plays would be unexpectedly 'sticky'. This setting allows to return to that behaviour.

See also *OLD_PLUGIN_CACHE_CLEARING*

ANSIBLE_PAGER

See also *PAGER*

Version Added
2.15

PAGER

See also *PAGER*

ANSIBLE_PARAMIKO_HOST_KEY_AUTO_ADD

See also *PARAMIKO_HOST_KEY_AUTO_ADD*

ANSIBLE_PARAMIKO_LOOK_FOR_KEYS

See also *PARAMIKO_LOOK_FOR_KEYS*

ANSIBLE_PERSISTENT_CONTROL_PATH_DIR

Path to socket to be used by the connection persistence system.

See also *PERSISTENT_CONTROL_PATH_DIR*

ANSIBLE_PERSISTENT_CONNECT_TIMEOUT

This controls how long the persistent connection will remain idle before it is destroyed.

See also *PERSISTENT_CONNECT_TIMEOUT*

ANSIBLE_PERSISTENT_CONNECT_RETRY_TIMEOUT

This controls the retry timeout for persistent connection to connect to the local domain socket.

See also *PERSISTENT_CONNECT_RETRY_TIMEOUT*

ANSIBLE_PERSISTENT_COMMAND_TIMEOUT

This controls the amount of time to wait for response from remote device before timing out persistent connection.

See also *PERSISTENT_COMMAND_TIMEOUT*

ANSIBLE_PLAYBOOK_DIR

A number of non-playbook CLIs have a `--playbook-dir` argument; this sets the default value for it.

See also *PLAYBOOK_DIR*

ANSIBLE_PLAYBOOK_VARS_ROOT

This sets which playbook dirs will be used as a root to process vars plugins, which includes finding host_vars/group_vars

See also *PLAYBOOK_VARS_ROOT*

ANSIBLE_PYTHON_MODULE_RLIMIT_NOFILE

Attempts to set RLIMIT_NOFILE soft limit to the specified value when executing Python modules (can speed up subprocess usage on Python 2.x. See https://bugs.python.org/issue11284). The value will be limited by the existing hard limit. Default value of 0 does not attempt to adjust existing system-defined limits.

See also *PYTHON_MODULE_RLIMIT_NOFILE*

ANSIBLE_RETRY_FILES_ENABLED

This controls whether a failed Ansible playbook should create a .retry file.

See also *RETRY_FILES_ENABLED*

ANSIBLE_RETRY_FILES_SAVE_PATH

This sets the path in which Ansible will save .retry files when a playbook fails and retry files are enabled.This file will be overwritten after each run with the list of failed hosts from all plays.

See also *RETRY_FILES_SAVE_PATH*

ANSIBLE_RUN_VARS_PLUGINS

This setting can be used to optimize vars_plugin usage depending on user's inventory size and play selection.

See also *RUN_VARS_PLUGINS*

ANSIBLE_SHOW_CUSTOM_STATS

This adds the custom stats set via the set_stats plugin to the default output

See also *SHOW_CUSTOM_STATS*

ANSIBLE_STRING_TYPE_FILTERS

This list of filters avoids 'type conversion' when templating variablesUseful when you want to avoid conversion into lists or dictionaries for JSON strings, for example.

See also *STRING_TYPE_FILTERS*

ANSIBLE_SYSTEM_WARNINGS

Allows disabling of warnings related to potential issues on the system running ansible itself (not on the managed hosts)These may include warnings about 3rd party packages or other conditions that should be resolved if possible.

See also *SYSTEM_WARNINGS*

ANSIBLE_RUN_TAGS

default list of tags to run in your plays, Skip Tags has precedence.

See also *TAGS_RUN*

ANSIBLE_SKIP_TAGS

default list of tags to skip in your plays, has precedence over Run Tags

See also *TAGS_SKIP*

ANSIBLE_TASK_TIMEOUT

Set the maximum time (in seconds) that a task can run for.If set to 0 (the default) there is no timeout.

See also *TASK_TIMEOUT*

ANSIBLE_WORKER_SHUTDOWN_POLL_COUNT

The maximum number of times to check Task Queue Manager worker processes to verify they have exited cleanly.After this limit is reached any worker processes still running will be terminated.This is for internal use only.

See also *WORKER_SHUTDOWN_POLL_COUNT*

ANSIBLE_WORKER_SHUTDOWN_POLL_DELAY

The number of seconds to sleep between polling loops when checking Task Queue Manager worker processes to verify they have exited cleanly.This is for internal use only.

See also *WORKER_SHUTDOWN_POLL_DELAY*

ANSIBLE_USE_PERSISTENT_CONNECTIONS

Toggles the use of persistence for connections.

See also *USE_PERSISTENT_CONNECTIONS*

ANSIBLE_VARS_ENABLED

Accept list for variable plugins that require it.

See also *VARIABLE_PLUGINS_ENABLED*

ANSIBLE_PRECEDENCE

Allows to change the group variable precedence merge order.

See also *VARIABLE_PRECEDENCE*

ANSIBLE_WIN_ASYNC_STARTUP_TIMEOUT

For asynchronous tasks in Ansible (covered in Asynchronous Actions and Polling), this is how long, in seconds, to wait for the task spawned by Ansible to connect back to the named pipe used on Windows systems. The default is 5 seconds. This can be too low on slower systems, or systems under heavy load.This is not the total time an async command can run for, but is a separate timeout to wait for an async command to start. The task will only start to be timed against its async_timeout once it has connected to the pipe, so the overall maximum duration the task can take will be extended by the amount specified here.

See also *WIN_ASYNC_STARTUP_TIMEOUT*

ANSIBLE_YAML_FILENAME_EXT

Check all of these extensions when looking for 'variable' files which should be YAML or JSON or vaulted versions of these.This affects vars_files, include_vars, inventory and vars plugins among others.

See also *YAML_FILENAME_EXTENSIONS*

ANSIBLE_NETCONF_SSH_CONFIG

This variable is used to enable bastion/jump host with netconf connection. If set to True the bastion/jump host ssh settings should be present in ~/.ssh/config file, alternatively it can be set to custom ssh configuration file path to read the bastion/jump host settings.

See also *NETCONF_SSH_CONFIG*

ANSIBLE_STRING_CONVERSION_ACTION

Action to take when a module parameter value is converted to a string (this does not affect variables). For string parameters, values such as '1.00', "['a', 'b',]", and 'yes', 'y', etc. will be converted by the YAML parser unless fully quoted.Valid options are 'error', 'warn', and 'ignore'.Since 2.8, this option defaults to 'warn' but will change to 'error' in 2.12.

See also *STRING_CONVERSION_ACTION*

ANSIBLE_VALIDATE_ACTION_GROUP_METADATA

A toggle to disable validating a collection's 'metadata' entry for a module_defaults action group. Metadata containing unexpected fields or value types will produce a warning when this is True.

See also *VALIDATE_ACTION_GROUP_METADATA*

ANSIBLE_VERBOSE_TO_STDERR

Force 'verbose' option to use stderr instead of stdout

See also *VERBOSE_TO_STDERR*

1.30 Controlling how Ansible behaves: precedence rules

To give you maximum flexibility in managing your environments, Ansible offers many ways to control how Ansible behaves: how it connects to managed nodes, how it works once it has connected. If you use Ansible to manage a large number of servers, network devices, and cloud resources, you may define Ansible behavior in several different places and pass that information to Ansible in several different ways. This flexibility is convenient, but it can backfire if you do not understand the precedence rules.

These precedence rules apply to any setting that can be defined in multiple ways (by configuration settings, command-line options, playbook keywords, variables).

1.30.1 Precedence categories

Ansible offers four sources for controlling its behavior. In order of precedence from lowest (most easily overridden) to highest (overrides all others), the categories are:

- Configuration settings
- Command-line options
- Playbook keywords
- Variables

Each category overrides any information from all lower-precedence categories. For example, a playbook keyword will override any configuration setting.

Within each precedence category, specific rules apply. However, generally speaking, 'last defined' wins and overrides any previous definitions.

Configuration settings

Configuration settings include both values from the `ansible.cfg` file and environment variables. Within this category, values set in configuration files have lower precedence. Ansible uses the first `ansible.cfg` file it finds, ignoring all others. Ansible searches for `ansible.cfg` in these locations in order:

- `ANSIBLE_CONFIG` (environment variable if set)
- `ansible.cfg` (in the current directory)
- `~/.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg`

Environment variables have a higher precedence than entries in `ansible.cfg`. If you have environment variables set on your control node, they override the settings in whichever `ansible.cfg` file Ansible loads. The value of any given environment variable follows normal shell precedence: the last value defined overwrites previous values.

Command-line options

Any command-line option will override any configuration setting.

When you type something directly at the command line, you may feel that your hand-crafted values should override all others, but Ansible does not work that way. Command-line options have low precedence - they override configuration only. They do not override playbook keywords, variables from inventory or variables from playbooks.

You can override all other settings from all other sources in all other precedence categories at the command line by *Using -e extra variables at the command line*, but that is not a command-line option, it is a way of passing a *variable*.

At the command line, if you pass multiple values for a parameter that accepts only a single value, the last defined value wins. For example, this *ad hoc task* will connect as `carol`, not as `mike`:

```
ansible -u mike -m ping myhost -u carol
```

Some parameters allow multiple values. In this case, Ansible will append all values from the hosts listed in inventory files inventory1 and inventory2:

```
ansible -i /path/inventory1 -i /path/inventory2 -m ping all
```

The help for each *command-line tool* lists available options for that tool.

Playbook keywords

Any *playbook keyword* will override any command-line option and any configuration setting.

Within playbook keywords, precedence flows with the playbook itself; the more specific wins against the more general:

- play (most general)
- blocks/includes/imports/roles (optional and can contain tasks and each other)
- tasks (most specific)

A simple example:

```
- hosts: all
  connection: ssh
  tasks:
    - name: This task uses ssh.
      ping:

    - name: This task uses paramiko.
      connection: paramiko
      ping:
```

In this example, the `connection` keyword is set to `ssh` at the play level. The first task inherits that value, and connects using `ssh`. The second task inherits that value, overrides it, and connects using `paramiko`. The same logic applies to blocks and roles as well. All tasks, blocks, and roles within a play inherit play-level keywords; any task, block, or role can override any keyword by defining a different value for that keyword within the task, block, or role.

Remember that these are KEYWORDS, not variables. Both playbooks and variable files are defined in YAML but they have different significance. Playbooks are the command or 'state description' structure for Ansible, variables are data we use to help make playbooks more dynamic.

Variables

Any variable will override any playbook keyword, any command-line option, and any configuration setting.

Variables that have equivalent playbook keywords, command-line options, and configuration settings are known as *Connection variables*. Originally designed for connection parameters, this category has expanded to include other core variables like the temporary directory and the python interpreter.

Connection variables, like all variables, can be set in multiple ways and places. You can define variables for hosts and groups in *inventory*. You can define variables for tasks and plays in `vars:` blocks in *playbooks*. However, they are still variables - they are data, not keywords or configuration settings. Variables that override playbook keywords, command-line options, and configuration settings follow the same rules of *variable precedence* as any other variables.

When set in a playbook, variables follow the same inheritance rules as playbook keywords. You can set a value for the play, then override it in a task, block, or role:

```
- hosts: cloud
  gather_facts: false
  become: true
  vars:
    ansible_become_user: admin
  tasks:
    - name: This task uses admin as the become user.
      dnf:
        name: some-service
        state: latest
    - block:
        - name: This task uses service-admin as the become user.
          # a task to configure the new service
        - name: This task also uses service-admin as the become user, defined in the␣
↪block.
          # second task to configure the service
      vars:
        ansible_become_user: service-admin
    - name: This task (outside of the block) uses admin as the become user again.
      service:
        name: some-service
        state: restarted
```

Variable scope: how long is a value available?

Variable values set in a playbook exist only within the playbook object that defines them. These 'playbook object scope' variables are not available to subsequent objects, including other plays.

Variable values associated directly with a host or group, including variables defined in inventory, by vars plugins, or using modules like set_fact and include_vars, are available to all plays. These 'host scope' variables are also available through the `hostvars[]` dictionary.

Using `-e` extra variables at the command line

To override all other settings in all other categories, you can use extra variables: `--extra-vars` or `-e` at the command line. Values passed with `-e` are variables, not command-line options, and they will override configuration settings, command-line options, and playbook keywords as well as variables set elsewhere. For example, this task will connect as `brian` not as `carol`:

```
ansible -u carol -e 'ansible_user=brian' -a whoami all
```

You must specify both the variable name and the value with `--extra-vars`.

1.31 YAML Syntax

This page provides a basic overview of correct YAML syntax, which is how Ansible playbooks (our configuration management language) are expressed.

We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.

You may also wish to read *Working with playbooks* at the same time to see how this is used in practice.

1.31.1 YAML Basics

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a "hash" or a "dictionary". So, we need to know how to write lists and dictionaries in YAML.

There's another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with `---` and end with `...`. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a `"- "` (a dash and a space):

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

A dictionary is represented in a simple `key: value` form (the colon must be followed by a space):

```
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

```
# Employee records
- martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

Dictionaries and lists can also be represented in an abbreviated form if you really want to:

```
---
martin: {name: Martin D'vloper, job: Developer, skill: Elite}
fruits: ['Apple', 'Orange', 'Strawberry', 'Mango']
```

These are called "Flow collections".

Ansible doesn't really use these too much, but you can also specify a *boolean value* (true/false) in several forms:

```
create_key: true
needs_agent: false
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

Use lowercase 'true' or 'false' for boolean values in dictionaries if you want to be compatible with default yamllint options.

Values can span multiple lines using | or >. Spanning multiple lines using a "Literal Block Scalar" | will include the newlines and any trailing spaces. Using a "Folded Block Scalar" > will fold newlines to spaces; it's used to make what would otherwise be a very long line easier to read and edit. In either case the indentation will be ignored. Examples are:

```
include_newlines: |
            exactly as you see
            will appear these three
            lines of poetry

fold_newlines: >
            this is really a
            single line of text
            despite appearances
```

While in the above > example all newlines are folded into spaces, there are two ways to enforce a newline to be kept:

```
fold_some_newlines: >
    a
    b

    c
    d
      e
    f
```

Alternatively, it can be enforced by including newline \n characters:

```
fold_same_newlines: "a b\nc d\n  e\nf\n"
```

Let's combine what we learned so far in an arbitrary YAML example. This really has nothing to do with Ansible, but will give you a feel for the format:

```
---
# An employee record
name: Martin D'vloper
```

(continues on next page)

(continued from previous page)

```
job: Developer
skill: Elite
employed: True
foods:
  - Apple
  - Orange
  - Strawberry
  - Mango
languages:
  perl: Elite
  python: Elite
  pascal: Lame
education: |
  4 GCSEs
  3 A-Levels
  BSc in the Internet of Things
```

That's all you really need to know about YAML to start writing *Ansible* playbooks.

1.31.2 Gotchas

While you can put just about anything into an unquoted scalar, there are some exceptions. A colon followed by a space (or newline) `": "` is an indicator for a mapping. A space followed by the pound sign `" #"` starts a comment.

Because of this, the following is going to result in a YAML syntax error:

```
foo: somebody said I should put a colon here: so I did

windows_drive: c:
```

. . . but this will work:

```
windows_path: c:\windows
```

You will want to quote hash values using colons followed by a space or the end of the line:

```
foo: 'somebody said I should put a colon here: so I did'

windows_drive: 'c:'
```

. . . and then the colon will be preserved.

Alternatively, you can use double quotes:

```
foo: "somebody said I should put a colon here: so I did"

windows_drive: "c:"
```

The difference between single quotes and double quotes is that in double quotes you can use escapes:

```
foo: "a \t TAB and a \n NEWLINE"
```

The list of allowed escapes can be found in the YAML Specification under "Escape Sequences" (YAML 1.1) or "Escape Characters" (YAML 1.2).

The following is invalid YAML:

```
foo: "an escaped \' single quote"
```

Further, Ansible uses "{{ var }}" for variables. If a value after a colon starts with a "{", YAML will think it is a dictionary, so you must quote it, like so:

```
foo: "{{ variable }}"
```

If your value starts with a quote the entire value must be quoted, not just part of it. Here are some additional examples of how to properly quote things:

```
foo: "{{ variable }}/additional/string/literal"
foo2: "{{ variable }}\\backslashes\\are\\also\\special\\characters"
foo3: "even if it's just a string literal it must all be quoted"
```

Not valid:

```
foo: "E:\\path\\"rest\\of\\path
```

In addition to `'` and `"` there are a number of characters that are special (or reserved) and cannot be used as the first character of an unquoted scalar: `[] {} > | * & ! % # ` @ ,`.

You should also be aware of `? : -`. In YAML, they are allowed at the beginning of a string if a non-space character follows, but YAML processor implementations differ, so it's better to use quotes.

In Flow Collections, the rules are a bit more strict:

```
a scalar in block mapping: this } is [ all , valid

flow mapping: { key: "you { should [ use , quotes here" }
```

Boolean conversion is helpful, but this can be a problem when you want a literal *yes* or other boolean values as a string. In these cases just use quotes:

```
non_boolean: "yes"
other_string: "False"
```

YAML converts certain strings into floating-point values, such as the string *1.0*. If you need to specify a version number (in a requirements.yml file, for example), you will need to quote the value if it looks like a floating-point value:

```
version: "1.0"
```

See also:

Working with playbooks
Learn what playbooks can do and how to write/run them.

YAMLLint
YAML Lint (online) helps you debug YAML syntax if you are having problems

GitHub examples directory
Complete playbook files from the github project source

Wikipedia YAML syntax reference
A good guide to YAML syntax

Mailing List
Questions? Help? Ideas? Stop by the list on Google Groups

Real-time chat
How to join Ansible chat channels (join #yaml for yaml-specific questions)

YAML 1.1 Specification
The Specification for YAML 1.1, which PyYAML and libyaml are currently implementing

YAML 1.2 Specification
For completeness, YAML 1.2 is the successor of 1.1

1.32 Python 3 Support

Ansible 2.5 and above work with Python 3. Previous to 2.5, using Python 3 was considered a tech preview. This topic discusses how to set up your controller and managed machines to use Python 3.

See :ref: *Control Node Requirements <control_node_requirements>* and :ref: *Managed Node Requirements <managed_node_requirements>* for the specific versions supported.

1.32.1 On the controller side

The easiest way to run **/usr/bin/ansible** under Python 3 is to install it with the Python3 version of pip. This will make the default **/usr/bin/ansible** run with Python3:

```
$ pip3 install ansible
$ ansible --version | grep "python version"
  python version = 3.10.5 (main, Jun 9 2022, 00:00:00) [GCC 12.1.1 20220507 (Red Hat 12.
↪1.1-1)] (/usr/bin/python)
```

If you are running Ansible *Running the devel branch from a clone* and want to use Python 3 with your source checkout, run your command through `python3`. For example:

```
$ source ./hacking/env-setup
$ python3 $(which ansible) localhost -m ping
$ python3 $(which ansible-playbook) sample-playbook.yml
```

Note: Individual Linux distribution packages may be packaged for Python2 or Python3. When running from distro packages you'll only be able to use Ansible with the Python version for which it was installed. Sometimes distros will provide a means of installing for several Python versions (through a separate package or through some commands that are run after install). You'll need to check with your distro to see if that applies in your case.

1.32.2 Using Python 3 on the managed machines with commands and playbooks

- Ansible will automatically detect and use Python 3 on many platforms that ship with it. To explicitly configure a Python 3 interpreter, set the `ansible_python_interpreter` inventory variable at a group or host level to the location of a Python 3 interpreter, such as **/usr/bin/python3**. The default interpreter path may also be set in `ansible.cfg`.

See also:

Interpreter Discovery for more information.

```
# Example inventory that makes an alias for localhost that uses Python3
localhost-py3 ansible_host=localhost ansible_connection=local ansible_python_
↪interpreter=/usr/bin/python3

# Example of setting a group of hosts to use Python3
[py3_hosts]
ubuntu16
fedora27

[py3_hosts:vars]
ansible_python_interpreter=/usr/bin/python3
```

See also:

How to build your inventory for more information.

- Run your command or playbook:

```
$ ansible localhost-py3 -m ping
$ ansible-playbook sample-playbook.yml
```

Note that you can also use the *-e* command line option to manually set the python interpreter when you run a command. This can be useful if you want to test whether a specific module or playbook has any bugs under Python 3. For example:

```
$ ansible localhost -m ping -e 'ansible_python_interpreter=/usr/bin/python3'
$ ansible-playbook sample-playbook.yml -e 'ansible_python_interpreter=/usr/bin/python3'
```

1.32.3 What to do if an incompatibility is found

We have spent several releases squashing bugs and adding new tests so that Ansible's core feature set runs under both Python 2 and Python 3. However, bugs may still exist in edge cases and many of the modules shipped with Ansible are maintained by the community and not all of those may be ported yet.

If you find a bug running under Python 3 you can submit a bug report on Ansible's GitHub project. Be sure to mention Python3 in the bug report so that the right people look at it.

If you would like to fix the code and submit a pull request on github, you can refer to *Ansible and Python 3* for information on how we fix common Python3 compatibility issues in the Ansible codebase.

1.33 Interpreter Discovery

Most Ansible modules that execute under a POSIX environment require a Python interpreter on the target host. Unless configured otherwise, Ansible will attempt to discover a suitable Python interpreter on each target host the first time a Python module is executed for that host.

To control the discovery behavior:

- for individual hosts and groups, use the `ansible_python_interpreter` inventory variable
- globally, use the `interpreter_python` key in the `[defaults]` section of `ansible.cfg`

Use one of the following values:

auto_legacy :
Detects the target OS platform, distribution, and version, then consults a table listing the correct Python interpreter and path for each platform/distribution/version. If an entry is found, and `/usr/bin/python` is absent, uses the discovered interpreter (and path). If an entry is found, and `/usr/bin/python` is present, uses `/usr/bin/python` and issues a warning. This exception provides temporary compatibility with previous versions of Ansible that always defaulted to `/usr/bin/python`, so if you have installed Python and other dependencies at `/usr/bin/python` on some hosts, Ansible will find and use them with this setting. If no entry is found, or the listed Python is not present on the target host, searches a list of common Python interpreter paths and uses the first one found; also issues a warning that future installation of another Python interpreter could alter the one chosen.

auto
[(default in 2.12)] Detects the target OS platform, distribution, and version, then consults a table listing the correct Python interpreter and path for each platform/distribution/version. If an entry is found, uses the discovered interpreter. If no entry is found, or the listed Python is not present on the target host, searches a list of common Python interpreter paths and uses the first one found; also issues a warning that future installation of another Python interpreter could alter the one chosen.

auto_legacy_silent
Same as `auto_legacy`, but does not issue warnings.

auto_silent
Same as `auto`, but does not issue warnings.

You can still set `ansible_python_interpreter` to a specific path at any variable level (for example, in host_vars, in vars files, in playbooks, and so on). Setting a specific path completely disables automatic interpreter discovery; Ansible always uses the path specified.

1.34 Releases and maintenance

This section describes release cycles, rules, and maintenance schedules for both Ansible community projects: the Ansible community package and `ansible-core`. The two projects have different versioning systems, maintenance structures, contents, and workflows.

Ansible community package	ansible-core
Uses new versioning (2.10, then 3.0.0)	Continues "classic Ansible" versioning (2.11, then 2.12)
Follows semantic versioning rules	Does not use semantic versioning
Maintains only one version at a time	Maintains latest version plus two older versions
Includes language, runtime, and selected Collections	Includes language, runtime, and builtin plugins
Developed and maintained in Collection repositories	Developed and maintained in ansible/ansible repository

Many community users install the Ansible community package. The Ansible community package offers the functionality that existed in Ansible 2.9, with more than 85 Collections containing thousands of modules and plugins. The `ansible-core` option is primarily for developers and users who want to install only the collections they need.

1.34.1 Release cycle overview

The two community releases are related - the release cycle follows this pattern:

1. Release of a new ansible-core major version, for example, ansible-core 2.11
   - New release of ansible-core and two prior versions are now maintained (in this case, ansible-base 2.10, Ansible 2.9)
   - Work on new features for ansible-core continues in the `devel` branch
2. Collection freeze (no new Collections or new versions of existing Collections) on the Ansible community package
3. Release candidate for Ansible community package, testing, additional release candidates as necessary
4. Release of a new Ansible community package major version based on the new ansible-core, for example, Ansible 4.0.0 based on ansible-core 2.11
   - Newest release of the Ansible community package is the only version now maintained
   - Work on new features continues in Collections
   - Individual Collections can make multiple minor and major releases
5. Minor releases of three maintained ansible-core versions every four weeks (2.11.1)
6. Minor releases of the single maintained Ansible community package version every four weeks (4.1.0)
7. Feature freeze on ansible-core
8. Release candidate for ansible-core, testing, additional release candidates as necessary
9. Release of the next ansible-core major version, cycle begins again

Ansible community package release cycle

The Ansible community team typically releases two major versions of the community package per year, on a flexible release cycle that trails the release of `ansible-core`. This cycle can be extended to allow for larger changes to be properly implemented and tested before a new release is made available. See *Ansible Roadmap* for upcoming release details. Between major versions, we release a new minor version of the Ansible community package every four weeks. Minor releases include new backwards-compatible features, modules and plugins, as well as bug fixes.

Starting with version 2.10, the Ansible community team guarantees maintenance for only one major community package release at a time. For example, when Ansible 4.0.0 gets released, the team will stop making new 3.x releases. Community members may maintain older versions if desired.

Note: Each Ansible EOL version may issue one final maintenance release at or shortly after the first release of the next version. When this happens, the final maintenance release is EOL at the date it releases.

Note: Older, unmaintained versions of the Ansible community package might contain unfixed security vulnerabilities (*CVEs*). If you are using a release of the Ansible community package that is no longer maintained, we strongly encourage you to upgrade as soon as possible to benefit from the latest features and security fixes.

Each major release of the Ansible community package accepts the latest released version of each included Collection and the latest released version of ansible-core. For specific schedules and deadlines, see the *Ansible Roadmap* for each version. Major releases of the Ansible community package can contain breaking changes in the modules and other plugins within the included Collections and in core features.

The Ansible community package follows semantic versioning rules. Minor releases of the Ansible community package accept only backwards-compatible changes in included Collections, that is, not Collections major releases. Collections must also use semantic versioning, so the Collection version numbers reflect this rule. For example, if Ansible 3.0.0 releases with community.general 2.0.0, then all minor releases of Ansible 3.x (such as Ansible 3.1.0 or Ansible 3.5.0) must include a 2.x release of community.general (such as 2.8.0 or 2.9.5) and not 3.x.x or later major releases.

Work in Collections is tracked within the individual Collection repositories.

You can refer to the *Ansible package porting guides* for tips on updating your playbooks to run on newer versions of Ansible. For Ansible 2.10 and later releases, you can install the Ansible package with `pip`. See *Installing Ansible* for details. You can download older Ansible releases from https://releases.ansible.com/ansible/.

Ansible community changelogs

This table links to the changelogs for each major Ansible release. These changelogs contain the dates and significant changes in each minor release.

Ansible Community Package Release	Status	Core version dependency
8.0.0	In development (unreleased)	2.15
7.x Changelogs	Current	2.14
6.x Changelogs	Unmaintained (end of life) after Ansible 6.7.0	2.13
5.x Changelogs	Unmaintained (end of life)	2.12
4.x Changelogs	Unmaintained (end of life)	2.11
3.x Changelogs	Unmaintained (end of life)	2.10
2.10 Changelogs	Unmaintained (end of life)	2.10

ansible-core release cycle

`ansible-core` is developed and released on a flexible release cycle. We can extend this cycle to properly implement and test larger changes before a new release is made available. See *ansible-core Roadmaps* for upcoming release details.

`ansible-core` has a graduated maintenance structure that extends to three major releases. For more information, read about the *Development and maintenance workflows* or see the chart in *ansible-core support matrix* for the degrees to which current releases are maintained.

Note: Older, unmaintained versions of `ansible-core` can contain unfixed security vulnerabilities (*CVEs*). If you are using a release of `ansible-core` that is no longer maintained, we strongly encourage you to upgrade as soon as possible to benefit from the latest features and security fixes. `ansible-core` maintenance continues for 3 releases. Thus the latest release receives security and general bug fixes when it is first released, security and critical bug fixes when the next `ansible-core` version is released, and **only** security fixes once the follow on to that version is released.

You can refer to the core_porting_guides for tips on updating your playbooks to run on newer versions of `ansible-core`.

You can install `ansible-core` with `pip`. See *Installing Ansible* for details.

`ansible-core` support matrix

This table links to the changelogs for each major `ansible-core` release. These changelogs contain the dates and significant changes in each minor release. Dates listed indicate the start date of the maintenance cycle.

Version	Support	End Of Life	Controller Python	Target Python / PowerShell
2.15	GA: 22 May 2023 Critical: 06 Nov 2023 Security: 20 May 2024	Nov 2024	Python 3.9 - 3.11	Python 2.7 Python 3.5 - 3.11 PowerShell 3 - 5.1
2.14	GA: 07 Nov 2022 Critical: 22 May 2023 Security: 06 Nov 2023	20 May 2024	Python 3.9 - 3.11	Python 2.7 Python 3.5 - 3.11 PowerShell 3 - 5.1
2.13	GA: 23 May 2022 Critical: 07 Nov 2022 Security: 22 May 2023	06 Nov 2023	Python 3.8 - 3.10	Python 2.7 Python 3.5 - 3.10 PowerShell 3 - 5.1
2.12	GA: 08 Nov 2021 Critical: 23 May 2022 Security: 07 Nov 2022	22 May 2023	Python 3.8 - 3.10	Python 2.6 Python 3.5 - 3.10 PowerShell 3 - 5.1
2.11	GA: 26 Apr 2021 Critical: 08 Nov 2021 Security: 23 May 2022	**EOL** 07 Nov 2022	Python 2.7 Python 3.5 - 3.9	Python 2.6 - 2.7 Python 3.5 - 3.9 PowerShell 3 - 5.1
2.10	GA: 13 Aug 2020 Critical: 26 Apr 2021 Security: 08 Nov 2021	**EOL** 23 May 2022	Python 2.7 Python 3.5 - 3.9	Python 2.6 - 2.7 Python 3.5 - 3.9 PowerShell 3 - 5.1
2.9	GA: 31 Oct 2019 Critical: 13 Aug 2020 Security: 26 Apr 2021	**EOL** 23 May 2022	Python 2.7 Python 3.5 - 3.8	Python 2.6 - 2.7 Python 3.5 - 3.8 PowerShell 3 - 5.1

1.34.2 Preparing for a new release

Feature freezes

During final preparations for a new release, core developers and maintainers focus on improving the release candidate, not on adding or reviewing new features. We may impose a feature freeze.

A feature freeze means that we delay new features and fixes unrelated to the pending release so we can create the new release as soon as possible.

Release candidates

We create at least one release candidate before each new major release of Ansible or `ansible-core`. Release candidates allow the Ansible community to try out new features, test existing playbooks on the release candidate, and report bugs or issues they find.

Ansible and `ansible-core` tag the first release candidate (RC1) which is usually scheduled to last five business days. If no major bugs or issues are identified during this period, the release candidate becomes the final release.

If there are major problems with the first candidate, the team and the community fix them and tag a second release candidate (RC2). This second candidate lasts for a shorter duration than the first. If no problems have been reported for an RC2 after two business days, the second release candidate becomes the final release.

If there are major problems in RC2, the cycle begins again with another release candidate and repeats until the maintainers agree that all major problems have been fixed.

1.34.3 Development and maintenance workflows

In between releases, the Ansible community develops new features, maintains existing functionality, and fixes bugs in `ansible-core` and in the collections included in the Ansible community package.

Ansible community package workflow

The Ansible community develops and maintains the features and functionality included in the Ansible community package in Collections repositories, with a workflow that looks like this:

- Developers add new features and bug fixes to the individual Collections, following each Collection's rules on contributing.
- Each new feature and each bug fix includes a changelog fragment describing the work.
- Release engineers create a minor release for the current version every four weeks to ensure that the latest bug fixes are available to users.
- At the end of the development period, the release engineers announce which Collections, and which major version of each included Collection, will be included in the next release of the Ansible community package. New Collections and new major versions may not be added after this, and the work of creating a new release begins.

We generally do not provide fixes for unmaintained releases of the Ansible community package, however, there can sometimes be exceptions for critical issues.

Some Collections are maintained by the Ansible team, some by Partner organizations, and some by community teams. For more information on adding features or fixing bugs in Ansible-maintained Collections, see *Contributing to Ansible-maintained Collections*.

ansible-core workflow

The Ansible community develops and maintains `ansible-core` on GitHub, with a workflow that looks like this:

- Developers add new features and bug fixes to the `devel` branch.
- Each new feature and each bug fix includes a changelog fragment describing the work.
- The development team backports bug fixes to one, two, or three stable branches, depending on the severity of the bug. They do not backport new features.
- Release engineers create a minor release for each maintained version every four weeks to ensure that the latest bug fixes are available to users.
- At the end of the development period, the release engineers impose a feature freeze and the work of creating a new release begins.

We generally do not provide fixes for unmaintained releases of `ansible-core`, however, there can sometimes be exceptions for critical issues.

For more information about adding features or fixing bugs in `ansible-core` see *The Ansible Development Cycle*.

Generating changelogs

We generate changelogs based on fragments. When creating new features for existing modules and plugins or fixing bugs, create a changelog fragment describing the change. A changelog entry is not needed for new modules or plugins. Details for those items will be generated from the module documentation.

To add changelog fragments to Collections in the Ansible community package, we recommend the antsibull-changelog utility.

To add changelog fragments for new features and bug fixes in `ansible-core`, see the *changelog examples and instructions* in the Community Guide.

1.34.4 Deprecation cycles

Sometimes we remove a feature, normally in favor of a reimplementation that we hope does a better job. To do this we have a deprecation cycle. First we mark a feature as 'deprecated'. This is normally accompanied with warnings to the user as to why we deprecated it, what alternatives they should switch to and when (which version) we are scheduled to remove the feature permanently.

Ansible community package deprecation cycle

Since Ansible is a package of individual collections, the deprecation cycle depends on the collection maintainers. We recommend the collection maintainers deprecate a feature in one Ansible major version and do not remove that feature for one year, or at least until the next major Ansible version. For example, deprecate the feature in 3.1.0 and do not remove the feature until 5.0.0 or 4.0.0 at the earliest. Collections should use semantic versioning, such that the major collection version cannot be changed within an Ansible major version. Therefore, the removal should not happen before the next major Ansible community package release. This is up to each collection maintainer and cannot be guaranteed.

ansible-core deprecation cycle

The deprecation cycle in `ansible-core` is normally across 4 feature releases (2.x. where the x marks a feature release). The feature is normally removed in the 4th release after we announce the deprecation. For example, something deprecated in 2.10 will be removed in 2.14. The tracking is tied to the number of releases, not the release numbering itself.

See also:

Committers Guidelines
Guidelines for Ansible core contributors and maintainers

Testing Strategies
Testing strategies

Ansible Community Guide
Community information and contributing

Development Mailing List
Mailing list for development topics

Real-time chat
How to join Ansible chat channels

1.35 Testing Strategies

1.35.1 Integrating Testing With Ansible Playbooks

Many times, people ask, "how can I best integrate testing with Ansible playbooks?" There are many options. Ansible is actually designed to be a "fail-fast" and ordered system, therefore it makes it easy to embed testing directly in Ansible playbooks. In this chapter, we'll go into some patterns for integrating tests of infrastructure and discuss the right level of testing that may be appropriate.

Note: This is a chapter about testing the application you are deploying, not the chapter on how to test Ansible modules during development. For that content, please hop over to the Development section.

By incorporating a degree of testing into your deployment workflow, there will be fewer surprises when code hits production and, in many cases, tests can be used in production to prevent failed updates from migrating across an entire installation. Since it's push-based, it's also very easy to run the steps on the localhost or testing servers. Ansible lets you insert as many checks and balances into your upgrade workflow as you would like to have.

1.35.2 The Right Level of Testing

Ansible resources are models of desired-state. As such, it should not be necessary to test that services are started, packages are installed, or other such things. Ansible is the system that will ensure these things are declaratively true. Instead, assert these things in your playbooks.

```
tasks:
  - ansible.builtin.service:
      name: foo
      state: started
      enabled: true
```

If you think the service may not be started, the best thing to do is request it to be started. If the service fails to start, Ansible will yell appropriately. (This should not be confused with whether the service is doing something functional, which we'll show more about how to do later).

1.35.3 Check Mode As A Drift Test

In the above setup, `--check` mode in Ansible can be used as a layer of testing as well. If running a deployment playbook against an existing system, using the `--check` flag to the *ansible* command will report if Ansible thinks it would have had to have made any changes to bring the system into a desired state.

This can let you know up front if there is any need to deploy onto the given system. Ordinarily, scripts and commands don't run in check mode, so if you want certain steps to execute in normal mode even when the `--check` flag is used, such as calls to the script module, disable check mode for those tasks:

```
roles:
  - webserver

tasks:
  - ansible.builtin.script: verify.sh
    check_mode: false
```

1.35.4 Modules That Are Useful for Testing

Certain playbook modules are particularly good for testing. Below is an example that ensures a port is open:

```
tasks:

  - ansible.builtin.wait_for:
      host: "{{ inventory_hostname }}"
      port: 22
    delegate_to: localhost
```

Here's an example of using the URI module to make sure a web service returns:

```
tasks:

  - action: uri url=https://www.example.com return_content=yes
    register: webpage

  - fail:
      msg: 'service is not happy'
    when: "'AWESOME' not in webpage.content"
```

It's easy to push an arbitrary script (in any language) on a remote host and the script will automatically fail if it has a non-zero return code:

```
tasks:

  - ansible.builtin.script: test_script1
  - ansible.builtin.script: test_script2 --parameter value --parameter2 value
```

If using roles (you should be, roles are great!), scripts pushed by the script module can live in the 'files/' directory of a role.

And the assert module makes it very easy to validate various kinds of truth:

```
tasks:

  - ansible.builtin.shell: /usr/bin/some-command --parameter value
    register: cmd_result

  - ansible.builtin.assert:
      that:
        - "'not ready' not in cmd_result.stderr"
        - "'gizmo enabled' in cmd_result.stdout"
```

Should you feel the need to test for the existence of files that are not declaratively set by your Ansible configuration, the 'stat' module is a great choice:

```
tasks:

  - ansible.builtin.stat:
      path: /path/to/something
    register: p

  - ansible.builtin.assert:
      that:
        - p.stat.exists and p.stat.isdir
```

As mentioned above, there's no need to check things like the return codes of commands. Ansible is checking them automatically. Rather than checking for a user to exist, consider using the user module to make it exist.

Ansible is a fail-fast system, so when there is an error creating that user, it will stop the playbook run. You do not have to check up behind it.

1.35.5 Testing Lifecycle

If writing some degree of basic validation of your application into your playbooks, they will run every time you deploy.

As such, deploying into a local development VM and a staging environment will both validate that things are according to plan ahead of your production deploy.

Your workflow may be something like this:

```
- Use the same playbook all the time with embedded tests in development
- Use the playbook to deploy to a staging environment (with the same playbooks) that␣
↪simulates production
- Run an integration test battery written by your QA team against staging
- Deploy to production, with the same integrated tests.
```

Something like an integration test battery should be written by your QA team if you are a production webservice. This would include things like Selenium tests or automated API tests and would usually not be something embedded into your Ansible playbooks.

However, it does make sense to include some basic health checks into your playbooks, and in some cases it may be possible to run a subset of the QA battery against remote nodes. This is what the next section covers.

1.35.6 Integrating Testing With Rolling Updates

If you have read into *Controlling where tasks run: delegation and local actions* it may quickly become apparent that the rolling update pattern can be extended, and you can use the success or failure of the playbook run to decide whether to add a machine into a load balancer or not.

This is the great culmination of embedded tests:

```
---

- hosts: webservers
  serial: 5

  pre_tasks:

    - name: take out of load balancer pool
      ansible.builtin.command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

  tasks:

    - ansible.builtin.include_role:
        name: "{{ item }}"
      loop:
        - common
        - webserver

    - name: run any notified handlers
      ansible.builtin.meta: flush_handlers

    - name: test the configuration
      ansible.builtin.include_role:
        name: apply_testing_checks

  post_tasks:

    - name: add back to load balancer pool
      ansible.builtin.command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1
```

Of course in the above, the "take out of the pool" and "add back" steps would be replaced with a call to an Ansible load balancer module or appropriate shell command. You might also have steps that use a monitoring module to start and end an outage window for the machine.

However, what you can see from the above is that tests are used as a gate – if the "apply_testing_checks" step is not performed, the machine will not go back into the pool.

Read the delegation chapter about "max_fail_percentage" and you can also control how many failing tests will stop a rolling update from proceeding.

This above approach can also be modified to run a step from a testing machine remotely against a machine:

```
---

- hosts: webservers
  serial: 5
```

(continues on next page)

(continued from previous page)

```
  pre_tasks:

    - name: take out of load balancer pool
      ansible.builtin.command: /usr/bin/take_out_of_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1

  roles:

     - common
     - webserver

  tasks:
     - ansible.builtin.script: /srv/qa_team/app_testing_script.sh --server {{ inventory_
↪hostname }}
       delegate_to: testing_server

  post_tasks:

    - name: add back to load balancer pool
      ansible.builtin.command: /usr/bin/add_back_to_pool {{ inventory_hostname }}
      delegate_to: 127.0.0.1
```

In the above example, a script is run from the testing server against a remote node prior to bringing it back into the pool.

In the event of a problem, fix the few servers that fail using Ansible's automatically generated retry file to repeat the deploy on just those servers.

1.35.7 Achieving Continuous Deployment

If desired, the above techniques may be extended to enable continuous deployment practices.

The workflow may look like this:

```
- Write and use automation to deploy local development VMs
- Have a CI system like Jenkins deploy to a staging environment on every code change
- The deploy job calls testing scripts to pass/fail a build on every deploy
- If the deploy job succeeds, it runs the same deploy playbook against production␣
↪inventory
```

Some Ansible users use the above approach to deploy a half-dozen or dozen times an hour without taking all of their infrastructure offline. A culture of automated QA is vital if you wish to get to this level.

If you are still doing a large amount of manual QA, you should still make the decision on whether to deploy manually as well, but it can still help to work in the rolling update patterns of the previous section and incorporate some basic health checks using modules like 'script', 'stat', 'uri', and 'assert'.

1.35.8 Conclusion

Ansible believes you should not need another framework to validate basic things of your infrastructure is true. This is the case because Ansible is an order-based system that will fail immediately on unhandled errors for a host, and prevent further configuration of that host. This forces errors to the top and shows them in a summary at the end of the Ansible run.

However, as Ansible is designed as a multi-tier orchestration system, it makes it very easy to incorporate tests into the end of a playbook run, either using loose tasks or roles. When used with rolling updates, testing steps can decide whether to put a machine back into a load balanced pool or not.

Finally, because Ansible errors propagate all the way up to the return code of the Ansible program itself, and Ansible by default runs in an easy push-based mode, Ansible is a great step to put into a build environment if you wish to use it to roll out systems as part of a Continuous Integration/Continuous Delivery pipeline, as is covered in sections above.

The focus should not be on infrastructure testing, but on application testing, so we strongly encourage getting together with your QA team and ask what sort of tests would make sense to run every time you deploy development VMs, and which sort of tests they would like to run against the staging environment on every deploy. Obviously at the development stage, unit tests are great too. But don't unit test your playbook. Ansible describes states of resources declaratively, so you don't have to. If there are cases where you want to be sure of something though, that's great, and things like stat/assert are great go-to modules for that purpose.

In all, testing is a very organizational and site-specific thing. Everybody should be doing it, but what makes the most sense for your environment will vary with what you are deploying and who is using it – but everyone benefits from a more robust and reliable deployment system.

See also:

Collection Index
: Browse existing collections, modules, and plugins

Working with playbooks
: An introduction to playbooks

Controlling where tasks run: delegation and local actions
: Delegation, useful for working with load balancers, clouds, and locally executed steps.

User Mailing List
: Have a question? Stop by the google group!

Real-time chat
: How to join Ansible chat channels

1.36 Frequently Asked Questions

Here are some commonly asked questions and their answers.

1.36.1 Where did all the modules go?

In July, 2019, we announced that collections would be the future of Ansible content delivery. A collection is a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. In Ansible 2.9 we added support for collections. In Ansible 2.10 we extracted most modules from the main ansible/ansible repository and placed them in collections. Collections may be maintained by the Ansible team, by the Ansible community, or by Ansible partners. The ansible/ansible repository now contains the code for basic features and functions, such as copying module code to managed nodes. This code is also known as `ansible-core` (it was briefly called `ansible-base` for version 2.10).

- To learn more about using collections, see *Using Ansible collections*.
- To learn more about developing collections, see *Developing collections*.
- To learn more about contributing to existing collections, see the individual collection repository for guidelines, or see *Contributing to Ansible-maintained Collections* to contribute to one of the Ansible-maintained collections.

1.36.2 Where did this specific module go?

IF you are searching for a specific module, you can check the runtime.yml file, which lists the first destination for each module that we extracted from the main ansible/ansible repository. Some modules have moved again since then. You can also search on Ansible Galaxy or ask on one of our *chat channels*.

1.36.3 How can I speed up Ansible on systems with slow disks?

Ansible may feel sluggish on systems with slow disks, such as Raspberry PI. See Ansible might be running slow if libyaml is not available for hints on how to improve this.

1.36.4 How can I set the PATH or any other environment variable for a task or entire play?

Setting environment variables can be done with the *environment* keyword. It can be used at the task or other levels in the play.

```
shell:
  cmd: date
environment:
  LANG=fr_FR.UTF-8
```

```
hosts: servers
environment:
  PATH: "{{ ansible_env.PATH }}:/thingy/bin"
  SOME: value
```

Note: starting in 2.0.1 the setup task from `gather_facts` also inherits the environment directive from the play, you might need to use the `|default` filter to avoid errors if setting this at play level.

1.36.5 How do I handle different machines needing different user accounts or ports to log in with?

Setting inventory variables in the inventory file is the easiest way.

For instance, suppose these hosts have different usernames and ports:

```
[webservers]
asdf.example.com  ansible_port=5000   ansible_user=alice
jkl.example.com   ansible_port=5001   ansible_user=bob
```

You can also dictate the connection type to be used, if you want:

```
[testcluster]
localhost           ansible_connection=local
/path/to/chroot1    ansible_connection=chroot
foo.example.com     ansible_connection=paramiko
```

You may also wish to keep these in group variables instead, or file them in a group_vars/<groupname> file. See the rest of the documentation for more information about how to organize variables.

1.36.6 How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?

Switch your default connection type in the configuration file to `ssh`, or use `-c ssh` to use Native OpenSSH for connections instead of the python paramiko library. In Ansible 1.2.1 and later, `ssh` will be used by default if OpenSSH is new enough to support ControlPersist as an option.

Paramiko is great for starting out, but the OpenSSH type offers many advanced options. You will want to run Ansible from a machine new enough to support ControlPersist, if you are using this connection type. You can still manage older clients. If you are using RHEL 6, CentOS 6, SLES 10 or SLES 11 the version of OpenSSH is still a bit old, so consider managing from a Fedora or openSUSE client even though you are managing older nodes, or just use paramiko.

We keep paramiko as the default as if you are first installing Ansible on these enterprise operating systems, it offers a better experience for new users.

1.36.7 How do I configure a jump host to access servers that I have no direct access to?

You can set a `ProxyCommand` in the `ansible_ssh_common_args` inventory variable. Any arguments specified in this variable are added to the sftp/scp/ssh command line when connecting to the relevant host(s). Consider the following inventory group:

```
[gatewayed]
foo ansible_host=192.0.2.1
bar ansible_host=192.0.2.2
```

You can create *group_vars/gatewayed.yml* with the following contents:

```
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q user@gateway.example.com"'
```

Ansible will append these arguments to the command line when trying to connect to any hosts in the group gatewayed. (These arguments are used in addition to any ssh_args from ansible.cfg, so you do not need to repeat global ControlPersist settings in ansible_ssh_common_args.)

Note that ssh -W is available only with OpenSSH 5.4 or later. With older versions, it's necessary to execute nc %h:%p or some equivalent command on the bastion host.

With earlier versions of Ansible, it was necessary to configure a suitable ProxyCommand for one or more hosts in ~/.ssh/config, or globally by setting ssh_args in ansible.cfg.

1.36.8 How do I get Ansible to notice a dead target in a timely manner?

You can add -o ServerAliveInterval=NumberOfSeconds with the ssh_args parameter in SSH connection plugin. Without this option, SSH and therefore Ansible will wait until the TCP connection times out. Another solution is to add ServerAliveInterval into your global SSH configuration. A good value for ServerAliveInterval is up to you to decide; keep in mind that ServerAliveCountMax=3 is the SSH default so any value you set will be tripled before terminating the SSH session.

1.36.9 How do I speed up run of ansible for servers from cloud providers (EC2, openstack,..)?

Don't try to manage a fleet of machines of a cloud provider from your laptop. Rather connect to a management node inside this cloud provider first and run Ansible from there.

1.36.10 How do I handle not having a Python interpreter at /usr/bin/python on a remote machine?

While you can write Ansible modules in any language, most Ansible modules are written in Python, including the ones central to letting Ansible work.

By default, Ansible assumes it can find a **/usr/bin/python** on your remote system that is either Python2, version 2.6 or higher or Python3, 3.5 or higher.

Setting the inventory variable ansible_python_interpreter on any host will tell Ansible to auto-replace the Python interpreter with that value instead. Thus, you can point to any Python you want on the system if **/usr/bin/python** on your system does not point to a compatible Python interpreter.

Some platforms may only have Python 3 installed by default. If it is not installed as **/usr/bin/python**, you will need to configure the path to the interpreter through ansible_python_interpreter. Although most core modules will work with Python 3, there may be some special purpose ones which do not or you may encounter a bug in an edge case. As a temporary workaround you can install Python 2 on the managed host and configure Ansible to use that Python through ansible_python_interpreter. If there's no mention in the module's documentation that the module requires Python 2, you can also report a bug on our bug tracker so that the incompatibility can be fixed in a future release.

Do not replace the shebang lines of your python modules. Ansible will do this for you automatically at deploy time.

Also, this works for ANY interpreter, for example ruby: ansible_ruby_interpreter, perl: ansible_perl_interpreter, and so on, so you can use this for custom modules written in any scripting language and control the interpreter location.

Keep in mind that if you put env in your module shebang line (#!/usr/bin/env <other>), this facility will be ignored so you will be at the mercy of the remote *$PATH*.

1.36.11 How do I handle the package dependencies required by Ansible package dependencies during Ansible installation ?

While installing Ansible, sometimes you may encounter errors such as *No package 'libffi' found* or *fatal error: Python.h: No such file or directory* These errors are generally caused by the missing packages, which are dependencies of the packages required by Ansible. For example, *libffi* package is dependency of *pynacl* and *paramiko* (Ansible -> paramiko -> pynacl -> libffi).

In order to solve these kinds of dependency issues, you might need to install required packages using the OS native package managers, such as *yum*, *dnf*, or *apt*, or as mentioned in the package installation guide.

Refer to the documentation of the respective package for such dependencies and their installation methods.

1.36.12 Common Platform Issues

What customer platforms does Red Hat support?

A number of them! For a definitive list please see this Knowledge Base article.

Running in a virtualenv

You can install Ansible into a virtualenv on the controller quite simply:

```
$ virtualenv ansible
$ source ./ansible/bin/activate
$ pip install ansible
```

If you want to run under Python 3 instead of Python 2 you may want to change that slightly:

```
$ virtualenv -p python3 ansible
$ source ./ansible/bin/activate
$ pip install ansible
```

If you need to use any libraries which are not available through pip (for instance, SELinux Python bindings on systems such as Red Hat Enterprise Linux or Fedora that have SELinux enabled), then you need to install them into the virtualenv. There are two methods:

- When you create the virtualenv, specify `--system-site-packages` to make use of any libraries installed in the system's Python:

  ```
  $ virtualenv ansible --system-site-packages
  ```

- Copy those files in manually from the system. For instance, for SELinux bindings you might do:

  ```
  $ virtualenv ansible --system-site-packages
  $ cp -r -v /usr/lib64/python3.*/site-packages/selinux/ ./py3-ansible/lib64/python3.
  ↪*/site-packages/
  $ cp -v /usr/lib64/python3.*/site-packages/*selinux*.so ./py3-ansible/lib64/python3.
  ↪*/site-packages/
  ```

Running on macOS as a controller

When executing Ansible on a system with macOS as a controller machine one might encounter the following error:

Error: +[__NSCFConstantString initialize] may have been in progress in another thread when fork() was called. We cannot safely call it or ignore it in the fork() child process. Crashing instead. Set a breakpoint on objc_initializeAfterForkError to debug. ERROR! A worker was found in a dead state

In general the recommended workaround is to set the following environment variable in your shell:

```
$ export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES
```

Running on macOS as a target

When managing a system with macOS Monterey 12, macOS Ventura 13 or above over SSH, the following error can occur:

Error: "eDSPermissionError" DS Error: -14120 (eDSPermissionError)

This is a good indication that *Allow full disk access for remote users* has not been enabled.

See also:

For more details, check out the official Apple user guide article.

Running on BSD

See also:

Managing BSD hosts with Ansible

Running on Solaris

By default, Solaris 10 and earlier run a non-POSIX shell which does not correctly expand the default tmp directory Ansible uses (~/.ansible/tmp). If you see module failures on Solaris machines, this is likely the problem. There are several workarounds:

- You can set remote_tmp to a path that will expand correctly with the shell you are using (see the plugin documentation for C shell, fish shell, and Powershell). For example, in the ansible config file you can set:

  ```
  remote_tmp=$HOME/.ansible/tmp
  ```

  In Ansible 2.5 and later, you can also set it per-host in inventory like this:

  ```
  solaris1 ansible_remote_tmp=$HOME/.ansible/tmp
  ```

- You can set *ansible_shell_executable* to the path to a POSIX compatible shell. For instance, many Solaris hosts have a POSIX shell located at /usr/xpg4/bin/sh so you can set this in inventory like so:

  ```
  solaris1 ansible_shell_executable=/usr/xpg4/bin/sh
  ```

  (bash, ksh, and zsh should also be POSIX compatible if you have any of those installed).

Running on z/OS

There are a few common errors that one might run into when trying to execute Ansible on z/OS as a target.

- Version 2.7.6 of python for z/OS will not work with Ansible because it represents strings internally as EBCDIC.

  To get around this limitation, download and install a later version of python for z/OS (2.7.13 or 3.6.1) that represents strings internally as ASCII. Version 2.7.13 is verified to work.

- When `pipelining = False` in */etc/ansible/ansible.cfg* then Ansible modules are transferred in binary mode through sftp however execution of python fails with

  Error: SyntaxError: Non-UTF-8 code starting with '\x83' in file /a/user1/.ansible/tmp/ansible-tmp-1548232945.35-274513842609025/AnsiballZ_stat.py on line 1, but no encoding declared; see https://python.org/dev/peps/pep-0263/ for details

  To fix it set `pipelining = True` in */etc/ansible/ansible.cfg*.

- Python interpret cannot be found in default location `/usr/bin/python` on target host.

  Error: /usr/bin/python: EDC5129I No such file or directory

  To fix this set the path to the python installation in your inventory like so:

  ```
  zos1 ansible_python_interpreter=/usr/lpp/python/python-2017-04-12-py27/python27/bin/
  ↪python
  ```

- Start of python fails with `The module libpython2.7.so was not found.`

  Error: EE3501S The module libpython2.7.so was not found.

  On z/OS, you must execute python from gnu bash. If gnu bash is installed at `/usr/lpp/bash`, you can fix this in your inventory by specifying an `ansible_shell_executable`:

  ```
  zos1 ansible_shell_executable=/usr/lpp/bash/bin/bash
  ```

Running under fakeroot

Some issues arise as `fakeroot` does not create a full nor POSIX compliant system by default. It is known that it will not correctly expand the default tmp directory Ansible uses (`~/.ansible/tmp`). If you see module failures, this is likely the problem. The simple workaround is to set `remote_tmp` to a path that will expand correctly (see documentation of the shell plugin you are using for specifics).

For example, in the ansible config file (or through environment variable) you can set:

```
remote_tmp=$HOME/.ansible/tmp
```

1.36.13 What is the best way to make content reusable/redistributable?

If you have not done so already, read all about "Roles" in the playbooks documentation. This helps you make playbook content self-contained, and works well with things like git submodules for sharing content with others.

If some of these plugin types look strange to you, see the API documentation for more details about ways Ansible can be extended.

1.36.14 Where does the configuration file live and what can I configure in it?

See *Configuring Ansible*.

1.36.15 How do I disable cowsay?

If cowsay is installed, Ansible takes it upon itself to make your day happier when running playbooks. If you decide that you would like to work in a professional cow-free environment, you can either uninstall cowsay, set `nocows=1` in `ansible.cfg`, or set the *`ANSIBLE_NOCOWS`* environment variable:

```
export ANSIBLE_NOCOWS=1
```

1.36.16 How do I see a list of all of the ansible_ variables?

Ansible by default gathers "facts" about the machines under management, and these facts can be accessed in playbooks and in templates. To see a list of all of the facts that are available about a machine, you can run the `setup` module as an ad hoc action:

```
ansible -m setup hostname
```

This will print out a dictionary of all of the facts that are available for that particular host. You might want to pipe the output to a pager.This does NOT include inventory variables or internal 'magic' variables. See the next question if you need more than just 'facts'.

1.36.17 How do I see all the inventory variables defined for my host?

By running the following command, you can see inventory variables for a host:

```
ansible-inventory --list --yaml
```

1.36.18 How do I see all the variables specific to my host?

To see all host specific variables, which might include facts and other sources:

```
ansible -m debug -a "var=hostvars['hostname']" localhost
```

Unless you are using a fact cache, you normally need to use a play that gathers facts first, for facts included in the task above.

1.36.19 How do I loop over a list of hosts in a group, inside of a template?

A pretty common pattern is to iterate over a list of hosts inside of a host group, perhaps to populate a template configuration file with a list of servers. To do this, you can just access the "$groups" dictionary in your template, like this:

```
{% for host in groups['db_servers'] %}
    {{ host }}
{% endfor %}
```

If you need to access facts about these hosts, for instance, the IP address of each hostname, you need to make sure that the facts have been populated. For example, make sure you have a play that talks to db_servers:

```
- hosts:  db_servers
  tasks:
    - debug: msg="doesn't matter what you do, just that they were talked to previously."
```

Then you can use the facts inside your template, like this:

```
{% for host in groups['db_servers'] %}
   {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
{% endfor %}
```

1.36.20 How do I access a variable name programmatically?

An example may come up where we need to get the ipv4 address of an arbitrary interface, where the interface to be used may be supplied through a role parameter or other input. Variable names can be built by adding strings together using "~", like so:

```
{{ hostvars[inventory_hostname]['ansible_' ~ which_interface]['ipv4']['address'] }}
```

The trick about going through hostvars is necessary because it's a dictionary of the entire namespace of variables. `inventory_hostname` is a magic variable that indicates the current host you are looping over in the host loop.

In the example above, if your interface names have dashes, you must replace them with underscores:

```
{{ hostvars[inventory_hostname]['ansible_' ~ which_interface | replace('_', '-') ]['ipv4
↪']['address'] }}
```

Also see *dynamic_variables*.

1.36.21 How do I access a group variable?

Technically, you don't, Ansible does not really use groups directly. Groups are labels for host selection and a way to bulk assign variables, they are not a first class entity, Ansible only cares about Hosts and Tasks.

That said, you could just access the variable by selecting a host that is part of that group, see *first_host_in_a_group* below for an example.

1.36.22 How do I access a variable of the first host in a group?

What happens if we want the ip address of the first webserver in the webservers group? Well, we can do that too. Note that if we are using dynamic inventory, which host is the 'first' may not be consistent, so you wouldn't want to do this unless your inventory is static and predictable. (If you are using AWX or the *Red Hat Ansible Automation Platform*, it will use database order, so this isn't a problem even if you are using cloud based inventory scripts).

Anyway, here's the trick:

```
{{ hostvars[groups['webservers'][0]]['ansible_eth0']['ipv4']['address'] }}
```

Notice how we're pulling out the hostname of the first machine of the webservers group. If you are doing this in a template, you could use the Jinja2 '#set' directive to simplify this, or in a playbook, you could also use set_fact:

```
- set_fact: headnode={{ groups['webservers'][0] }}

- debug: msg={{ hostvars[headnode].ansible_eth0.ipv4.address }}
```

Notice how we interchanged the bracket syntax for dots – that can be done anywhere.

1.36.23 How do I copy files recursively onto a target host?

The `copy` module has a recursive parameter. However, take a look at the `synchronize` module if you want to do something more efficient for a large number of files. The `synchronize` module wraps rsync. See the module index for info on both of these modules.

1.36.24 How do I access shell environment variables?

On controller machine : Access existing variables from controller use the `env` lookup plugin. For example, to access the value of the HOME environment variable on the management machine:

```
---
# ...
  vars:
     local_home: "{{ lookup('env','HOME') }}"
```

On target machines : Environment variables are available through facts in the `ansible_env` variable:

```
{{ ansible_env.HOME }}
```

If you need to set environment variables for TASK execution, see *Setting the remote environment* in the Advanced Playbooks section. There are several ways to set environment variables on your target machines. You can use the template, replace, or lineinfile modules to introduce environment variables into files. The exact files to edit vary depending on your OS and distribution and local configuration.

1.36.25 How do I generate encrypted passwords for the user module?

Ansible ad hoc command is the easiest option:

```
ansible all -i localhost, -m debug -a "msg={{ 'mypassword' | password_hash('sha512',
↪'mysecretsalt') }}"
```

The `mkpasswd` utility that is available on most Linux systems is also a great option:

```
mkpasswd --method=sha-512
```

If this utility is not installed on your system (for example, you are using macOS) then you can still easily generate these passwords using Python. First, ensure that the Passlib password hashing library is installed:

```
pip install passlib
```

Once the library is ready, SHA512 password values can then be generated as follows:

```
python -c "from passlib.hash import sha512_crypt; import getpass; print(sha512_crypt.
↪using(rounds=5000).hash(getpass.getpass()))"
```

Use the integrated *Hashing and encrypting strings and passwords* to generate a hashed version of a password. You shouldn't put plaintext passwords in your playbook or host_vars; instead, use *Using encrypted variables and files* to encrypt sensitive data.

In OpenBSD, a similar option is available in the base system called `encrypt (1)`

1.36.26 Ansible allows dot notation and array notation for variables. Which notation should I use?

The dot notation comes from Jinja and works fine for variables without special characters. If your variable contains dots (.), colons (:), or dashes (-), if a key begins and ends with two underscores, or if a key uses any of the known public attributes, it is safer to use the array notation. See *Using Variables* for a list of the known public attributes.

```
item[0]['checksum:md5']
item['section']['2.1']
item['region']['Mid-Atlantic']
It is {{ temperature['Celsius']['-3'] }} outside.
```

Also array notation allows for dynamic variable composition, see *dynamic_variables*.

Another problem with 'dot notation' is that some keys can cause problems because they collide with attributes and methods of python dictionaries.

- Example of incorrect syntax when `item` is a dictionary:

```
item.update
```

This variant causes a syntax error because `update()` is a Python method for dictionaries.

- Example of correct syntax:

```
item['update']
```

1.36.27 When is it unsafe to bulk-set task arguments from a variable?

You can set all of a task's arguments from a dictionary-typed variable. This technique can be useful in some dynamic execution scenarios. However, it introduces a security risk. We do not recommend it, so Ansible issues a warning when you do something like this:

```
#...
vars:
  usermod_args:
    name: testuser
    state: present
    update_password: always
tasks:
- user: '{{ usermod_args }}'
```

This particular example is safe. However, constructing tasks like this is risky because the parameters and values passed to `usermod_args` could be overwritten by malicious values in the `host facts` on a compromised target machine. To mitigate this risk:

- set bulk variables at a level of precedence greater than `host facts` in the order of precedence found in *Variable precedence: Where should I put a variable?* (the example above is safe because play vars take precedence over facts)
- disable the *INJECT_FACTS_AS_VARS* configuration setting to prevent fact values from colliding with variables (this will also disable the original warning)

1.36.28 Can I get training on Ansible?

Yes! See our services page for information on our services and training offerings. Email info@ansible.com for further details.

We also offer free web-based training classes on a regular basis. See our webinar page for more info on upcoming webinars.

1.36.29 Is there a web interface / REST API / GUI?

Yes! The open-source web interface is Ansible AWX. The supported Red Hat product that makes Ansible even more powerful and easy to use is *Red Hat Ansible Automation Platform*.

1.36.30 How do I keep secret data in my playbook?

If you would like to keep secret data in your Ansible content and still share it publicly or keep things in source control, see *Using encrypted variables and files*.

If you have a task that you don't want to show the results or command given to it when using -v (verbose) mode, the following task or playbook attribute can be useful:

```
- name: secret task
  shell: /usr/bin/do_something --value={{ secret_value }}
  no_log: True
```

This can be used to keep verbose output but hide sensitive information from others who would otherwise like to be able to see the output.

The no_log attribute can also apply to an entire play:

```
- hosts: all
  no_log: True
```

Though this will make the play somewhat difficult to debug. It's recommended that this be applied to single tasks only, once a playbook is completed. Note that the use of the no_log attribute does not prevent data from being shown when debugging Ansible itself through the *ANSIBLE_DEBUG* environment variable.

1.36.31 When should I use {{ }}? Also, how to interpolate variables or dynamic variable names

A steadfast rule is 'always use `{{ }}` except when `when:`'. Conditionals are always run through Jinja2 as to resolve the expression, so `when:`, `failed_when:` and `changed_when:` are always templated and you should avoid adding `{{ }}`.

In most other cases you should always use the brackets, even if previously you could use variables without specifying (like `loop` or `with_` clauses), as this made it hard to distinguish between an undefined variable and a string.

Another rule is 'moustaches don't stack'. We often see this:

```
{{ somevar_{{other_var}} }}
```

The above DOES NOT WORK as you expect, if you need to use a dynamic variable use the following as appropriate:

```
{{ hostvars[inventory_hostname]['somevar_' ~ other_var] }}
```

For 'non host vars' you can use the vars lookup plugin:

```
{{ lookup('vars', 'somevar_' ~ other_var) }}
```

To determine if a keyword requires `{{ }}` or even supports templating, use `ansible-doc -t keyword <name>`, this will return documentation on the keyword including a `template` field with the values `explicit` (requires `{{ }}`), `implicit` (assumes `{{ }}`, so no needed) or `static` (no templating supported, all characters will be interpreted literally)

1.36.32 How do I get the original ansible_host when I delegate a task?

As the documentation states, connection variables are taken from the `delegate_to` host so `ansible_host` is overwritten, but you can still access the original through `hostvars`:

```
original_host: "{{ hostvars[inventory_hostname]['ansible_host'] }}"
```

This works for all overridden connection variables, like `ansible_user`, `ansible_port`, and so on.

1.36.33 How do I fix 'protocol error: filename does not match request' when fetching a file?

Since release `7.9p1` of OpenSSH there is a bug in the SCP client that can trigger this error on the Ansible controller when using SCP as the file transfer mechanism:

Error: failed to transfer file to /tmp/ansible/file.txtrnprotocol error: filename does not match request

In these releases, SCP tries to validate that the path of the file to fetch matches the requested path. The validation fails if the remote filename requires quotes to escape spaces or non-ascii characters in its path. To avoid this error:

- **Use SFTP instead of SCP by setting `scp_if_ssh` to `smart` (which tries SFTP first) or to `False`. You can do this in one of four ways:**
  - Rely on the default setting, which is `smart` - this works if `scp_if_ssh` is not explicitly set anywhere
  - Set a *host variable* or *group variable* in inventory: `ansible_scp_if_ssh: False`
  - Set an environment variable on your control node: `export ANSIBLE_SCP_IF_SSH=False`
  - Pass an environment variable when you run Ansible: `ANSIBLE_SCP_IF_SSH=smart ansible-playbook`
  - Modify your `ansible.cfg` file: add `scp_if_ssh=False` to the `[ssh_connection]` section
- **If you must use SCP, set the `-T` arg to tell the SCP client to ignore path validation. You can do this in one of three ways:**
  - Set a *host variable* or *group variable*: `ansible_scp_extra_args=-T`,
  - Export or pass an environment variable: `ANSIBLE_SCP_EXTRA_ARGS=-T`
  - Modify your `ansible.cfg` file: add `scp_extra_args=-T` to the `[ssh_connection]` section

Note: If you see an `invalid argument` error when using `-T`, then your SCP client is not performing filename validation and will not trigger this error.

1.36.34 Does Ansible support multiple factor authentication 2FA/MFA/biometrics/finterprint/usbkey/OTP/…

No, Ansible is designed to execute multiple tasks against multiple targets, minimizing user interaction. As with most automation tools, it is not compatible with interactive security systems designed to handle human interaction. Most of these systems require a secondary prompt per target, which prevents scaling to thousands of targets. They also tend to have very short expiration periods so it requires frequent reauthorization, also an issue with many hosts and/or a long set of tasks.

In such environments we recommend securing around Ansible's execution but still allowing it to use an 'automation user' that does not require such measures. With AWX or the *Red Hat Ansible Automation Platform*, administrators can set up RBAC access to inventory, along with managing credentials and job execution.

1.36.35 The 'validate' option is not enough for my needs, what do I do?

Many Ansible modules that create or update files have a `validate` option that allows you to abort the update if the validation command fails. This uses the temporary file Ansible creates before doing the final update. In many cases this does not work since the validation tools for the specific application require either specific names, multiple files or some other factor that is not present in this simple feature.

For these cases you have to handle the validation and restoration yourself. The following is a simple example of how to do this with block/rescue and backups, which most file based modules also support:

```
- name: update config and backout if validation fails
  block:
     - name: do the actual update, works with copy, lineinfile and any action that␣
↪allows for `backup`.
       template: src=template.j2 dest=/x/y/z backup=yes moreoptions=stuff
       register: updated

    - name: run validation, this will change a lot as needed. We assume it returns an␣
↪error when not passing, use `failed_when` if otherwise.
      shell: run_validation_commmand
      become: true
      become_user: requiredbyapp
      environment:
        WEIRD_REQUIREMENT: 1
 rescue:
    - name: restore backup file to original, in the hope the previous configuration was␣
↪working.
      copy:
         remote_src: true
         dest: /x/y/z
         src: "{{ updated['backup_file'] }}"
 always:
    - name: We choose to always delete backup, but could copy or move, or only delete in␣
↪rescue.
      file:
         path: "{{ updated['backup_file'] }}"
         state: absent
```

1.36.36 Why does the `regex_search` filter return *None* instead of an empty string?

Until the jinja2 2.10 release, Jinja was only able to return strings, but Ansible needed Python objects in some cases. Ansible uses `safe_eval` and only sends strings that look like certain types of Python objects through this function. With `regex_search` that does not find a match, the result (`None`) is converted to the string "None" which is not useful in non-native jinja2.

The following example of a single templating action shows this behavior:

```
{{ 'ansible' | regex_search('foobar') }}
```

This example does not result in a Python `None`, so Ansible historically converted it to "" (empty string).

The native jinja2 functionality actually allows us to return full Python objects, that are always represented as Python objects everywhere, and as such the result of a single templating action with `regex_search` can result in the Python `None`.

Note: Native jinja2 functionality is not needed when `regex_search` is used as an intermediate result that is then compared to the jinja2 `none` test.

```
{{ 'ansible' | regex_search('foobar') is none }}
```

1.36.37 How do I submit a change to the documentation?

Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs README viewable on GitHub. Thanks!

1.36.38 What is the difference between `ansible.legacy` and `ansible.builtin` collections?

Neither is a real collection. They are virtually constructed by the core engine (synthetic collections).

The `ansible.builtin` collection only refers to plugins that ship with `ansible-core`.

The `ansible.legacy` collection is a superset of `ansible.builtin` (you can reference the plugins from builtin through `ansible.legacy`). You also get the ability to add 'custom' plugins in the *configured paths and adjacent directories*, with the ability to override the builtin plugins that have the same name.

Also, `ansible.legacy` is what you get by default when you do not specify an FQCN. So this:

```
- shell: echo hi
```

Is really equivalent to:

```
- ansible.legacy.shell: echo hi
```

Though, if you do not override the `shell` module, you can also just write it as `ansible.builtin.shell`, since legacy will resolve to the builtin collection.

1.36.39 I don't see my question here

If you have not found an answer to your questions, you can ask on one of our mailing lists or chat channels. For instructions on subscribing to a list or joining a chat channel, see *Communicating with the Ansible community*.

See also:

Working with playbooks
An introduction to playbooks

Ansible tips and tricks
Tips and tricks for playbooks

User Mailing List
Have a question? Stop by the google group!

1.37 Glossary

The following is a list (and re-explanation) of term definitions used elsewhere in the Ansible documentation.

Consult the documentation home page for the full documentation and to see the terms in context, but this should be a good resource to check your knowledge of Ansible's components and understand how they fit together. It's something you might wish to read for review or when a term comes up on the mailing list.

Action
An action is a part of a task that specifies which of the modules to run and which arguments to pass to that module. Each task can have only one action, but it may also have other parameters.

Ad Hoc
Refers to running Ansible to perform some quick command, using **`/usr/bin/ansible`**, rather than the *orchestration* language, which is **`/usr/bin/ansible-playbook`**. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a *playbook* and playbooks can also glue lots of other operations together.

Ansible (the package)
A software package (Python, deb, rpm, and so on) that contains ansible-core and a select group of collections. Playbooks that worked with Ansible 2.9 should still work with the Ansible 2.10 package. See the `ansible-<version>.build` file in the release-specific directory at ansible-build-data for a list of collections included in Ansible, as well as the included `ansible-core` version.

ansible-base
Used only for 2.10. The installable package (RPM/Python/Deb package) generated from the ansible/ansible repository. See `ansible-core`.

ansible-core
Name used starting with 2.11. The installable package (RPM/Python/Deb package) generated from the ansible/ansible repository. Contains the command-line tools and the code for basic features and functions, such as copying module code to managed nodes. The `ansible-core` package includes a few modules and plugins and allows you to add others by installing collections.

Ansible Galaxy
An online distribution server for finding and sharing Ansible community content, sometimes referred to as community Galaxy. Also, the command-line utility that lets users install individual Ansible Collections, for example `ansible-galaxy collection install community.crypto`.

Async
Refers to a task that is configured to run in the background rather than waiting for completion. If you have a long process that would run longer than the SSH timeout, it would make sense to launch that task in async mode. Async modes can poll for completion every so many seconds or can be configured to "fire and forget", in which case Ansible will not even check on the task again; it will just kick it off and proceed to future steps. Async modes work with both **`/usr/bin/ansible`** and **`/usr/bin/ansible-playbook`**.

Callback Plugin
Refers to some user-written code that can intercept results from Ansible and do something with them. Some supplied examples in the GitHub project perform custom logging, send email, or even play sound effects.

Check Mode
Refers to running Ansible with the `--check` option, which does not make any changes on the remote systems, but only outputs the changes that might occur if the command ran without this flag. This is analogous to so-called "dry run" modes in other systems, though the user should be warned that this does not take into account unexpected command failures or cascade effects (which is true of similar modes in other systems). Use this to get an idea of what might happen, but do not substitute it for a good staging environment.

Collection
A packaging format for bundling and distributing Ansible content, including plugins, roles, modules, and more.

Collections release independent of other collections or `ansible-core` so features can be available sooner to users. Some collections are packaged with Ansible (version 2.10 or later). You can install other collections (or other versions of collections) with `ansible-galaxy collection install <namespace.collection>`.

Collection name
The second part of a Fully Qualified Collection Name. The collection name divides the collection namespace and usually reflects the function of the collection content. For example, the `cisco` namespace might contain `cisco.ios`, `cisco.aci`, and `cisco.nxos`, with content for managing the different network devices maintained by Cisco.

community.general (collection)
A special collection managed by the Ansible Community Team containing all the modules and plugins which shipped in Ansible 2.9 that do not have their own dedicated Collection. See community.general on Galaxy.

community.network (collection)
Similar to `community.general`, focusing on network content. community.network on Galaxy.

Connection Plugin
By default, Ansible talks to remote machines through pluggable libraries. Ansible uses native OpenSSH (*SSH (Native)*) or a Python implementation called *paramiko*. OpenSSH is preferred if you are using a recent version, and also enables some features like Kerberos and jump hosts. This is covered in the getting started section. There are also other connection types like `accelerate` mode, which must be bootstrapped over one of the SSH-based connection types but is very fast, and local mode, which acts on the local system. Users can also write their own connection plugins.

Conditionals
A conditional is an expression that evaluates to true or false that decides whether a given task is executed on a given machine or not. Ansible's conditionals are powered by the 'when' statement, which are discussed in the *Working with playbooks*.

Declarative
An approach to achieving a task that uses a description of the final state rather than a description of the sequence of steps necessary to achieve that state. For a real world example, a declarative specification of a task would be: "put me in California". Depending on your current location, the sequence of steps to get you to California may vary, and if you are already in California, nothing at all needs to be done. Ansible's Resources are declarative; it figures out the steps needed to achieve the final state. It also lets you know whether or not any steps needed to be taken to get to the final state.

Diff Mode
A `--diff` flag can be passed to Ansible to show what changed on modules that support it. You can combine it with `--check` to get a good 'dry run'. File diffs are normally in unified diff format.

Distribution server
A server, such as Ansible Galaxy or Red Hat Automation Hub where you can distribute your collections and allow others to access these collections. See *Distributing collections* for a list of distribution server types. Some Ansible features are only available on certain distribution servers.

Executor
A core software component of Ansible that is the power behind **`/usr/bin/ansible`** directly – and corresponds to the invocation of each task in a *playbook*. The Executor is something Ansible developers may talk about, but it's not really user land vocabulary.

Facts
Facts are simply things that are discovered about remote nodes. While they can be used in *playbooks* and templates just like variables, facts are things that are inferred, rather than set. Facts are automatically discovered by Ansible when running plays by executing the internal setup module on the remote nodes. You never have to call the setup module explicitly, it just runs, but it can be disabled to save time if it is not needed or you can tell ansible to collect only a subset of the full facts through the `gather_subset:` option. For the convenience of users who are switching from other configuration management systems, the fact module will also pull in facts

from the **ohai** and **facter** tools if they are installed. These are fact libraries from Chef and Puppet, respectively. (These may also be disabled through `gather_subset:`)

Filter Plugin
A filter plugin is something that most users will never need to understand. These allow for the creation of new *Jinja2* filters, which are more or less only of use to people who know what Jinja2 filters are. If you need them, you can learn how to write them in the *API docs section*.

Forks
Ansible talks to remote nodes in parallel and the level of parallelism can be set either by passing `--forks` or editing the default in a configuration file. The default is a very conservative five (5) forks, though if you have a lot of RAM, you can easily set this to a value like 50 for increased parallelism.

Fully Qualified Collection Name (FQCN)
The full definition of a module, plugin, or role hosted within a collection, in the form <namespace.collection.content_name>. Allows a Playbook to refer to a specific module or plugin from a specific source in an unambiguous manner, for example, `community.grafana.grafana_dashboard`. The FQCN is required when you want to specify the exact source of a plugin. For example, if multiple collections contain a module plugin called `user`, the FQCN specifies which one to use for a given task. When you have multiple collections installed, the FQCN is always the explicit and authoritative indicator of which collection to search for the correct plugin for each task.

Gather Facts (Boolean)
Facts are mentioned above. Sometimes when running a multi-play *playbook*, it is desirable to have some plays that don't bother with fact computation if they aren't going to need to utilize any of these values. Setting `gather_facts:  False` on a playbook allows this implicit fact gathering to be skipped.

Globbing
Globbing is a way to select lots of hosts based on wildcards, rather than the name of the host specifically, or the name of the group they are in. For instance, it is possible to select `ww*` to match all hosts starting with `www`. This concept is pulled directly from **Func**, one of Michael DeHaan's (an Ansible Founder) earlier projects. In addition to basic globbing, various set operations are also possible, such as 'hosts in this group and not in another group', and so on.

Group
A group consists of several hosts assigned to a pool that can be conveniently targeted together, as well as given variables that they share in common.

Group Vars
The `group_vars/` files are files that live in a directory alongside an inventory file, with an optional filename named after each group. This is a convenient place to put variables that are provided to a given group, especially complex data structures, so that these variables do not have to be embedded in the *inventory* file or *playbook*.

Handlers
Handlers are just like regular tasks in an Ansible *playbook* (see *Tasks*) but are only run if the Task contains a `notify` keyword and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

Host
A host is simply a remote machine that Ansible manages. They can have individual variables assigned to them, and can also be organized in groups. All hosts have a name they can be reached at (which is either an IP address or a domain name) and, optionally, a port number, if they are not to be accessed on the default SSH port.

Host Specifier
Each *Play* in Ansible maps a series of *tasks* (which define the role, purpose, or orders of a system) to a set of systems.

This `hosts:` keyword in each play is often called the hosts specifier.

It may select one system, many systems, one or more groups, or even some hosts that are in one group and explicitly not in another.

Host Vars
Just like *Group Vars*, a directory alongside the inventory file named `host_vars/` can contain a file named after each hostname in the inventory file, in *YAML* format. This provides a convenient place to assign variables to the host without having to embed them in the *inventory* file. The Host Vars file can also be used to define complex data structures that can't be represented in the inventory file.

Idempotency
An operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions.

Includes
The idea that *playbook* files (which are nothing more than lists of *plays*) can include other lists of plays, and task lists can externalize lists of *tasks* in other files, and similarly with *handlers*. Includes can be parameterized, which means that the loaded file can pass variables. For instance, an included play for setting up a WordPress blog may take a parameter called `user` and that play could be included more than once to create a blog for both `alice` and `bob`.

Inventory
A file (by default, Ansible uses a simple INI format) that describes *Hosts* and *Groups* in Ansible. Inventory can also be provided through an *Inventory Script* (sometimes called an "External Inventory Script").

Inventory Script
A very simple program (or a complicated one) that looks up *hosts*, *group* membership for hosts, and variable information from an external resource – whether that be a SQL database, a CMDB solution, or something like LDAP. This concept was adapted from Puppet (where it is called an "External Nodes Classifier") and works more or less exactly the same way.

Jinja2
Jinja2 is the preferred templating language of Ansible's template module. It is a very simple Python template language that is generally readable and easy to write.

JSON
Ansible uses JSON for return data from remote modules. This allows modules to be written in any language, not just Python.

Keyword
The main expressions that make up Ansible, which apply to playbook objects (Play, Block, Role and Task). For example 'vars:' is a keyword that lets you define variables in the scope of the playbook object it is applied to.

Lazy Evaluation
In general, Ansible evaluates any variables in *playbook* content at the last possible second, which means that if you define a data structure that data structure itself can define variable values within it, and everything "just works" as you would expect. This also means variable strings can include other variables inside of those strings.

Library
A collection of modules made available to **`/usr/bin/ansible`** or an Ansible *playbook*.

Limit Groups
By passing `--limit somegroup` to **`ansible`** or **`ansible-playbook`**, the commands can be limited to a subset of *hosts*. For instance, this can be used to run a *playbook* that normally targets an entire set of servers to one particular server.

Local Action
This keyword is an alias for `delegate_to: localhost`. Used when you want to redirect an action from the remote to execute on the controller itself.

Local Connection
By using `connection: local` in a *playbook*, or passing `-c local` to **`/usr/bin/ansible`**, this indicates that we are executing a local fork instead of executing on the remote machine. You probably want `local_action` or `delegate_to: localhost` instead as this ONLY changes the connection and no other context for execution.

Lookup Plugin
A lookup plugin is a way to get data into Ansible from the outside world. Lookup plugins are an extension of Jinja2 and can be accessed in templates, for example, `{{ lookup('file','/path/to/file') }}`. These are how such things as `with_items`, are implemented. There are also lookup plugins like `file` which loads data from a file and ones for querying environment variables, DNS text records, or key value stores.

Loops
Generally, Ansible is not a programming language. It prefers to be more declarative, though various constructs like `loop` allow a particular task to be repeated for multiple items in a list. Certain modules, like yum and apt, actually take lists directly, and can install all packages given in those lists within a single transaction, dramatically speeding up total time to configuration, so they can be used without loops.

Modules
Modules are the units of work that Ansible ships out to remote machines. Modules are kicked off by either **`/usr/bin/ansible`** or **`/usr/bin/ansible-playbook`** (where multiple tasks use lots of different modules in conjunction). Modules can be implemented in any language, including Perl, Bash, or Ruby – but can take advantage of some useful communal library code if written in Python. Modules just have to return *JSON*. Once modules are executed on remote machines, they are removed, so no long running daemons are used. Ansible refers to the collection of available modules as a *library*.

Multi-Tier
The concept that IT systems are not managed one system at a time, but by interactions between multiple systems and groups of systems in well defined orders. For instance, a web server may need to be updated before a database server and pieces on the web server may need to be updated after *THAT* database server and various load balancers and monitoring servers may need to be contacted. Ansible models entire IT topologies and workflows rather than looking at configuration from a "one system at a time" perspective.

Namespace
The first part of a fully qualified collection name, the namespace usually reflects a functional content category. Example: in `cisco.ios.ios_config`, `cisco` is the namespace. Namespaces are reserved and distributed by Red Hat at Red Hat's discretion. Many, but not all, namespaces will correspond with vendor names. See Galaxy namespaces on the Galaxy docsite for namespace requirements.

Notify
The act of a *task* registering a change event and informing a *handler* task that another *action* needs to be run at the end of the *play*. If a handler is notified by multiple tasks, it will still be run only once. Handlers are run in the order they are listed, not in the order that they are notified.

Orchestration
Many software automation systems use this word to mean different things. Ansible uses it as a conductor would conduct an orchestra. A datacenter or cloud architecture is full of many systems, playing many parts – web servers, database servers, maybe load balancers, monitoring systems, continuous integration systems, and so on. In performing any process, it is necessary to touch systems in particular orders, often to simulate rolling updates or to deploy software correctly. Some system may perform some steps, then others, then previous systems already processed may need to perform more steps. Along the way, emails may need to be sent or web services contacted. Ansible orchestration is all about modeling that kind of process.

paramiko
By default, Ansible manages machines over SSH. The library that Ansible uses by default to do this is a Python-powered library called paramiko. The paramiko library is generally fast and easy to manage, though users who want to use Kerberos or Jump Hosts may wish to switch to a native SSH binary such as OpenSSH by specifying the connection type in their *playbooks*, or using the `-c ssh` flag.

Playbooks

Playbooks are the language by which Ansible orchestrates, configures, administers, or deploys systems. They are called playbooks partially because it's a sports analogy, and it's supposed to be fun using them. They aren't workbooks :)

Plays

A *playbook* is a list of plays. A play is minimally a mapping between a set of *hosts* selected by a host specifier (usually chosen by *groups* but sometimes by hostname *globs*) and the *tasks* which run on those hosts to define the role that those systems will perform. There can be one or many plays in a playbook.

Pull Mode

By default, Ansible runs in *push mode*, which allows it very fine-grained control over when it talks to each system. Pull mode is provided for when you would rather have nodes check in every N minutes on a particular schedule. It uses a program called **ansible-pull** and can also be set up (or reconfigured) using a push-mode *playbook*. Most Ansible users use push mode, but pull mode is included for variety and the sake of having choices.

ansible-pull works by checking configuration orders out of git on a crontab and then managing the machine locally, using the *local connection* plugin.

Pulp 3 Galaxy

A self-hosted distribution server based on the GalaxyNG codebase, based on Pulp version 3. Use it to find and share your own curated set of content. You can access your content with the `ansible-galaxy collection` command.

Push Mode

Push mode is the default mode of Ansible. In fact, it's not really a mode at all – it's just how Ansible works when you aren't thinking about it. Push mode allows Ansible to be fine-grained and conduct nodes through complex orchestration processes without waiting for them to check in.

Register Variable

The result of running any *task* in Ansible can be stored in a variable for use in a template or a conditional statement. The keyword used to define the variable is called `register`, taking its name from the idea of registers in assembly programming (though Ansible will never feel like assembly programming). There are an infinite number of variable names you can use for registration.

Resource Model

Ansible modules work in terms of resources. For instance, the file module will select a particular file and ensure that the attributes of that resource match a particular model. As an example, we might wish to change the owner of `/etc/motd` to `root` if it is not already set to `root`, or set its mode to `0644` if it is not already set to `0644`. The resource models are *idempotent* meaning change commands are not run unless needed, and Ansible will bring the system back to a desired state regardless of the actual state – rather than you having to tell it how to get to the state.

Roles

Roles are units of organization in Ansible. Assigning a role to a group of *hosts* (or a set of *groups*, or *host patterns*, and so on) implies that they should implement a specific behavior. A role may include applying certain variable values, certain *tasks*, and certain *handlers* – or just one or more of these things. Because of the file structure associated with a role, roles become redistributable units that allow you to share behavior among *playbooks* – or even with other users.

Rolling Update

The act of addressing a number of nodes in a group N at a time to avoid updating them all at once and bringing the system offline. For instance, in a web topology of 500 nodes handling very large volume, it may be reasonable to update 10 or 20 machines at a time, moving on to the next 10 or 20 when done. The `serial:` keyword in an Ansible *playbooks* control the size of the rolling update pool. The default is to address the batch size all at once, so this is something that you must opt-in to. OS configuration (such as making sure config files are correct) does not typically have to use the rolling update model, but can do so if desired.

Serial

See also:

Rolling Update

Sudo
Ansible does not require root logins, and since it's daemonless, definitely does not require root level daemons (which can be a security concern in sensitive environments). Ansible can log in and perform many operations wrapped in a sudo command, and can work with both password-less and password-based sudo. Some operations that don't normally work with sudo (like scp file transfer) can be achieved with Ansible's copy, template, and fetch modules while running in sudo mode.

SSH (Native)
Native OpenSSH as an Ansible transport is specified with `-c ssh` (or a config file, or a keyword in the *playbook*) and can be useful if wanting to login through Kerberized SSH or using SSH jump hosts, and so on. In 1.2.1, `ssh` will be used by default if the OpenSSH binary on the control machine is sufficiently new. Previously, Ansible selected `paramiko` as a default. Using a client that supports `ControlMaster` and `ControlPersist` is recommended for maximum performance – if you don't have that and don't need Kerberos, jump hosts, or other features, `paramiko` is a good choice. Ansible will warn you if it doesn't detect ControlMaster/ControlPersist capability.

Tags
Ansible allows tagging resources in a *playbook* with arbitrary keywords, and then running only the parts of the playbook that correspond to those keywords. For instance, it is possible to have an entire OS configuration, and have certain steps labeled `ntp`, and then run just the `ntp` steps to reconfigure the time server information on a remote host.

Task
Playbooks exist to run tasks. Tasks combine an *action* (a module and its arguments) with a name and optionally some other keywords (like *looping keywords*). *Handlers* are also tasks, but they are a special kind of task that do not run unless they are notified by name when a task reports an underlying change on a remote system.

Tasks
A list of *Task*.

Templates
Ansible can easily transfer files to remote systems but often it is desirable to substitute variables in other files. Variables may come from the *inventory* file, *Host Vars*, *Group Vars*, or *Facts*. Templates use the *Jinja2* template engine and can also include logical constructs like loops and if statements.

Transport
Ansible uses :term:`Connection Plugins` to define types of available transports. These are simply how Ansible will reach out to managed systems. Transports included are *paramiko*, *ssh* (using OpenSSH), and *local*.

When
An optional conditional statement attached to a *task* that is used to determine if the task should run or not. If the expression following the `when:` keyword evaluates to false, the task will be ignored.

Vars (Variables)
As opposed to *Facts*, variables are names of values (they can be simple scalar values – integers, booleans, strings) or complex ones (dictionaries/hashes, lists) that can be used in templates and *playbooks*. They are declared things, not things that are inferred from the remote system's current state or nature (which is what Facts are).

YAML
Ansible does not want to force people to write programming language code to automate infrastructure, so Ansible uses YAML to define *playbook* configuration languages and also variable files. YAML is nice because it has a minimum of syntax and is very clean and easy for people to skim. It is a good data format for configuration files and humans, but also machine readable. Ansible's usage of YAML stemmed from Michael DeHaan's first use of it inside of Cobbler around 2006. YAML is fairly popular in the dynamic language community and the format has libraries available for serialization in many languages (Python, Perl, Ruby, and so on).

See also:

Frequently Asked Questions
Frequently asked questions

Working with playbooks
An introduction to playbooks

Ansible tips and tricks
Tips and tricks for playbooks

User Mailing List
Have a question? Stop by the google group!

Real-time chat
How to join Ansible chat channels

1.38 Ansible Reference: Module Utilities

This page documents utilities intended to be helpful when writing Ansible modules in Python.

1.38.1 AnsibleModule

To use this functionality, include `from ansible.module_utils.basic import AnsibleModule` in your module.

`class ansible.module_utils.basic.`**`AnsibleModule`**`(`*argument_spec*, *bypass_checks=False*, *no_log=False*, *mutually_exclusive=None*, *required_together=None*, *required_one_of=None*, *add_file_common_args=False*, *supports_check_mode=False*, *required_if=None*, *required_by=None*`)`

Common code for quickly building an ansible module in Python (although you can write modules with anything that can return JSON).

See *Developing modules* for a general introduction and *Ansible module architecture* for more detailed explanation.

`add_path_info``(`*kwargs*`)`

for results that are files, supplement the info about the file in the return path with stats about the file path.

`atomic_move``(`*src*, *dest*, *unsafe_writes=False*`)`

atomically move src to dest, copying attributes from dest, returns true on success it uses os.rename to ensure this as it is an atomic operation, rest of the function is to work around limitations, corner cases and ensure selinux context is saved if possible

`backup_local``(`*fn*`)`

make a date-marked backup of the specified file, return True or False on success or failure

`boolean``(`*arg*`)`

Convert the argument to a boolean

`digest_from_file``(`*filename*, *algorithm*`)`

Return hex digest of local file for a digest_method specified by name, or None if file is not present.

`exit_json``(`***kwargs*`)`

return from the module, without error

`fail_json`(*msg*, ***kwargs*)

return from the module, with an error message

`find_mount_point`(*path*)

Takes a path and returns it's mount point

Parameters

`path` – a string type with a filesystem path

Returns

the path to the mount point as a text type

`get_bin_path`(*arg*, *required=False*, *opt_dirs=None*)

Find system executable in PATH.

Parameters

- `arg` – The executable to find.
- `required` – if executable is not found and required is `True`, fail_json
- `opt_dirs` – optional list of directories to search in addition to `PATH`

Returns

if found return full path; otherwise return None

`is_executable`(*path*)

is the given path executable?

Parameters

`path` – The path of the file to check.

Limitations:

- Does not account for FSACLs.
- Most times we really want to know "Can the current user execute this file". This function does not tell us that, only if any execute bit is set.

`is_special_selinux_path`(*path*)

Returns a tuple containing (True, selinux_context) if the given path is on a NFS or other 'special' fs mount point, otherwise the return will be (False, None).

`load_file_common_arguments`(*params*, *path=None*)

many modules deal with files, this encapsulates common options that the file module accepts such that it is directly available to all modules and they can share code.

Allows to overwrite the path/dest module argument by providing path.

`md5`(*filename*)

Return MD5 hex digest of local file using digest_from_file().

Do not use this function unless you have no other choice for:

1) Optional backwards compatibility
2) Compatibility with a third party protocol

This function will not work on systems complying with FIPS-140-2.

Most uses of this function can use the module.sha1 function instead.

`preserved_copy`(*src*, *dest*)

Copy a file with preserved ownership, permissions and context

`run_command`(*args*, *check_rc=False*, *close_fds=True*, *executable=None*, *data=None*, *binary_data=False*, *path_prefix=None*, *cwd=None*, *use_unsafe_shell=False*, *prompt_regex=None*, *environ_update=None*, *umask=None*, *encoding='utf-8'*, *errors='surrogate_or_strict'*, *expand_user_and_vars=True*, *pass_fds=None*, *before_communicate_callback=None*, *ignore_invalid_cwd=True*, *handle_exceptions=True*)

Execute a command, returns rc, stdout, and stderr.

Parameters

`args` – is the command to run * If args is a list, the command will be run with shell=False. * If args is a string and use_unsafe_shell=False it will split args to a list and run with shell=False * If args is a string and use_unsafe_shell=True it runs with shell=True.

Kw check_rc

Whether to call fail_json in case of non zero RC. Default False

Kw close_fds

See documentation for subprocess.Popen(). Default True

Kw executable

See documentation for subprocess.Popen(). Default None

Kw data

If given, information to write to the stdin of the command

Kw binary_data

If False, append a newline to the data. Default False

Kw path_prefix

If given, additional path to find the command in. This adds to the PATH environment variable so helper commands in the same directory can also be found

Kw cwd

If given, working directory to run the command inside

Kw use_unsafe_shell

See *args* parameter. Default False

Kw prompt_regex

Regex string (not a compiled regex) which can be used to detect prompts in the stdout which would otherwise cause the execution to hang (especially if no input data is specified)

Kw environ_update

dictionary to *update* environ variables with

Kw umask

Umask to be used when running the command. Default None

Kw encoding

Since we return native strings, on python3 we need to know the encoding to use to transform from bytes to text. If you want to always get bytes back, use encoding=None. The default is "utf-8". This does not affect transformation of strings given as args.

Kw errors

Since we return native strings, on python3 we need to transform stdout and stderr from bytes to text. If the bytes are undecodable in the `encoding` specified, then use this error handler to deal with them. The default is `surrogate_or_strict` which means that the bytes will be decoded using the surrogateescape error handler if available (available on all python3 versions we support) otherwise a UnicodeError traceback will be raised. This does not affect transformations of strings given as args.

Kw expand_user_and_vars
When `use_unsafe_shell=False` this argument dictates whether ~ is expanded in paths and environment variables are expanded before running the command. When `True` a string such as `$SHELL` will be expanded regardless of escaping. When `False` and `use_unsafe_shell=False` no path or variable expansion will be done.

Kw pass_fds
When running on Python 3 this argument dictates which file descriptors should be passed to an underlying `Popen` constructor. On Python 2, this will set `close_fds` to False.

Kw before_communicate_callback
This function will be called after `Popen` object will be created but before communicating to the process. (`Popen` object will be passed to callback as a first argument)

Kw ignore_invalid_cwd
This flag indicates whether an invalid `cwd` (non-existent or not a directory) should be ignored or should raise an exception.

Kw handle_exceptions
This flag indicates whether an exception will be handled inline and issue a failed_json or if the caller should handle it.

Returns
A 3-tuple of return code (integer), stdout (native string), and stderr (native string). On python2, stdout and stderr are both byte strings. On python3, stdout and stderr are text strings converted according to the encoding and errors parameters. If you want byte strings on python3, use encoding=None to turn decoding to text off.

`sha1`(*filename*)
Return SHA1 hex digest of local file using digest_from_file().

`sha256`(*filename*)
Return SHA-256 hex digest of local file using digest_from_file().

1.38.2 Basic

To use this functionality, include `import ansible.module_utils.basic` in your module.

`ansible.module_utils.basic.`**`get_all_subclasses`**(*cls*)
Deprecated: Use ansible.module_utils.common._utils.get_all_subclasses instead

`ansible.module_utils.basic.`**`get_platform`**()
Deprecated Use `platform.system()` directly.
Returns
Name of the platform the module is running on in a native string
Returns a native string that labels the platform ("Linux", "Solaris", etc). Currently, this is the result of calling `platform.system()`.

`ansible.module_utils.basic.`**`heuristic_log_sanitize`**(*data*, *no_log_values=None*)
Remove strings that look like passwords from log messages

`ansible.module_utils.basic.`**`load_platform_subclass`**(*cls*, **args*, ***kwargs*)
Deprecated: Use ansible.module_utils.common.sys_info.get_platform_subclass instead

1.38.3 Argument Spec

Classes and functions for validating parameters against an argument spec.

ArgumentSpecValidator

class `ansible.module_utils.common.arg_spec.`**`ArgumentSpecValidator`**(*argument_spec*, *mutually_exclusive=None*, *required_together=None*, *required_one_of=None*, *required_if=None*, *required_by=None*)

Argument spec validation class

Creates a validator based on the `argument_spec` that can be used to validate a number of parameters using the `validate()` method.

Parameters

- **`argument_spec`** (*`dict[str, dict]`*) – Specification of valid parameters and their type. May include nested argument specs.
- **`mutually_exclusive`** (*`list[str] or list[list[str]]`*) – List or list of lists of terms that should not be provided together.
- **`required_together`** (*`list[list[str]]`*) – List of lists of terms that are required together.
- **`required_one_of`** (*`list[list[str]]`*) – List of lists of terms, one of which in each list is required.
- **`required_if`** (*`list`*) – List of lists of `[parameter, value, [parameters]]` where one of `[parameters]` is required if `parameter == value`.
- **`required_by`** (*`dict[str, list[str]]`*) – Dictionary of parameter names that contain a list of parameters required by each key in the dictionary.

`validate`(*parameters*, **args*, ***kwargs*)

Validate `parameters` against argument spec.

Error messages in the `ValidationResult` may contain no_log values and should be sanitized with `sanitize_keys()` before logging or displaying.

Parameters

`parameters` (*`dict[str, dict]`*) – Parameters to validate against the argument spec

Returns

`ValidationResult` containing validated parameters.

Simple Example

```
argument_spec = {
    'name': {'type': 'str'},
    'age': {'type': 'int'},
}

parameters = {
    'name': 'bo',
    'age': '42',
}
```

(continues on next page)

(continued from previous page)

```
validator = ArgumentSpecValidator(argument_spec)
result = validator.validate(parameters)

if result.error_messages:
    sys.exit("Validation failed: {0}".format(", ".join(result.
↪error_messages))

valid_params = result.validated_parameters
```

ValidationResult

class `ansible.module_utils.common.arg_spec.`**`ValidationResult`**(*parameters*)

Result of argument spec validation.

This is the object returned by *`ArgumentSpecValidator.validate()`* containing the validated parameters and any errors.

Parameters

`parameters` (*`dict`*) – Terms to be validated and coerced to the correct type.

`errors`

`AnsibleValidationErrorMultiple` containing all *`AnsibleValidationError`* objects if there were any failures during validation.

property validated_parameters

Validated and coerced parameters.

property unsupported_parameters

`set` of unsupported parameter names.

property error_messages

`list` of all error messages from each exception in *`errors`*.

Parameters

`ansible.module_utils.common.parameters.`**`DEFAULT_TYPE_VALIDATORS`**

`dict` of type names, such as `'str'`, and the default function used to check that type, *`check_type_str()`* in this case.

`ansible.module_utils.common.parameters.`**`env_fallback`**(**args*, ***kwargs*)

Load value from environment variable

`ansible.module_utils.common.parameters.`**`remove_values`**(*value*, *no_log_strings*)

Remove strings in `no_log_strings` from value.

If value is a container type, then remove a lot more.

Use of `deferred_removals` exists, rather than a pure recursive solution, because of the potential to hit the maximum recursion depth when dealing with large amounts of data (see issue #24560).

`ansible.module_utils.common.parameters.`**`sanitize_keys`**(*obj*, *no_log_strings*, *ignore_keys=frozenset({})*)

Sanitize the keys in a container object by removing `no_log` values from key names.

This is a companion function to the *remove_values()* function. Similar to that function, we make use of `deferred_removals` to avoid hitting maximum recursion depth in cases of large data structures.

Parameters

- **obj** – The container object to sanitize. Non-container objects are returned unmodified.
- **no_log_strings** – A set of string values we do not want logged.
- **ignore_keys** – A set of string values of keys to not sanitize.

Returns

An object with sanitized keys.

Validation

Standalone functions for validating various parameter types.

`ansible.module_utils.common.validation.`**check_missing_parameters**(*parameters*, *required_parameters=None*)

This is for checking for required params when we can not check via argspec because we need more information than is simply given in the argspec.

Raises `TypeError` if any required parameters are missing

Parameters

- **parameters** – Dictionary of parameters
- **required_parameters** – List of parameters to look for in the given parameters.

Returns

Empty list or raises `TypeError` if the check fails.

`ansible.module_utils.common.validation.`**check_mutually_exclusive**(*terms*, *parameters*, *options_context=None*)

Check mutually exclusive terms against argument parameters

Accepts a single list or list of lists that are groups of terms that should be mutually exclusive with one another

Parameters

- **terms** – List of mutually exclusive parameters
- **parameters** – Dictionary of parameters
- **options_context** – List of strings of parent key names if `terms` are in a sub spec.

Returns

Empty list or raises `TypeError` if the check fails.

`ansible.module_utils.common.validation.`**check_required_arguments**(*argument_spec*, *parameters*, *options_context=None*)

Check all parameters in argument_spec and return a list of parameters that are required but not present in parameters.

Raises `TypeError` if the check fails

Parameters

- **argument_spec** – Argument spec dictionary containing all parameters and their specification
- **parameters** – Dictionary of parameters

- **options_context** – List of strings of parent key names if `argument_spec` are in a sub spec.

Returns
Empty list or raises `TypeError` if the check fails.

`ansible.module_utils.common.validation.`**`check_required_by`**`(`*requirements*, *parameters*, *options_context=None*`)`

For each key in requirements, check the corresponding list to see if they exist in parameters.

Accepts a single string or list of values for each key.

Parameters

- **requirements** – Dictionary of requirements
- **parameters** – Dictionary of parameters
- **options_context** – List of strings of parent key names if `requirements` are in a sub spec.

Returns
Empty dictionary or raises `TypeError` if the

`ansible.module_utils.common.validation.`**`check_required_if`**`(`*requirements*, *parameters*, *options_context=None*`)`

Check parameters that are conditionally required

Raises `TypeError` if the check fails

Parameters
requirements – List of lists specifying a parameter, value, parameters required when the given parameter is the specified value, and optionally a boolean indicating any or all parameters are required.

Example

```
required_if=[
    ['state', 'present', ('path',), True],
    ['someint', 99, ('bool_param', 'string_param')],
]
```

Parameters

- **parameters** – Dictionary of parameters
- **options_context** – List of strings of parent key names if `requirements` are in a sub spec.

Returns

Empty list or raises `TypeError` if the check fails. The results attribute of the exception contains a list of dictionaries. Each dictionary is the result of evaluating each item in requirements. Each return dictionary contains the following keys:

key missing
List of parameters that are required but missing

key requires
'any' or 'all'

key parameter
Parameter name that has the requirement

key value
Original value of the parameter

key requirements
Original required parameters

Example

```
[
    {
        'parameter': 'someint',
        'value': 99
        'requirements': ('bool_param', 'string_param'),
        'missing': ['string_param'],
        'requires': 'all',
    }
]
```

`ansible.module_utils.common.validation.`**`check_required_one_of`**(*terms*, *parameters*, *options_context=None*)

Check each list of terms to ensure at least one exists in the given module parameters

Accepts a list of lists or tuples

Parameters

- **`terms`** – List of lists of terms to check. For each list of terms, at least one is required.
- **`parameters`** – Dictionary of parameters
- **`options_context`** – List of strings of parent key names if `terms` are in a sub spec.

Returns
Empty list or raises `TypeError` if the check fails.

`ansible.module_utils.common.validation.`**`check_required_together`**(*terms*, *parameters*, *options_context=None*)

Check each list of terms to ensure every parameter in each list exists in the given parameters.

Accepts a list of lists or tuples.

Parameters

- **`terms`** – List of lists of terms to check. Each list should include parameters that are all required when at least one is specified in the parameters.
- **`parameters`** – Dictionary of parameters
- **`options_context`** – List of strings of parent key names if `terms` are in a sub spec.

Returns
Empty list or raises `TypeError` if the check fails.

`ansible.module_utils.common.validation.`**`check_type_bits`**(*value*)

Convert a human-readable string bits value to bits in integer.

Example: `check_type_bits('1Mb')` returns integer 1048576.

Raises `TypeError` if unable to covert the value.

`ansible.module_utils.common.validation.`**`check_type_bool`**`(`*value*`)`

Verify that the value is a bool or convert it to a bool and return it.

Raises `TypeError` if unable to convert to a bool

Parameters

`value` – String, int, or float to convert to bool. Valid booleans include: '1', 'on', 1, '0', 0, 'n', 'f', 'false', 'true', 'y', 't', 'yes', 'no', 'off'

Returns

Boolean True or False

`ansible.module_utils.common.validation.`**`check_type_bytes`**`(`*value*`)`

Convert a human-readable string value to bytes

Raises `TypeError` if unable to covert the value

`ansible.module_utils.common.validation.`**`check_type_dict`**`(`*value*`)`

Verify that value is a dict or convert it to a dict and return it.

Raises `TypeError` if unable to convert to a dict

Parameters

`value` – Dict or string to convert to a dict. Accepts `k1=v2, k2=v2`.

Returns

value converted to a dictionary

`ansible.module_utils.common.validation.`**`check_type_float`**`(`*value*`)`

Verify that value is a float or convert it to a float and return it

Raises `TypeError` if unable to convert to a float

Parameters

`value` – float, int, str, or bytes to verify or convert and return.

Returns

float of given value.

`ansible.module_utils.common.validation.`**`check_type_int`**`(`*value*`)`

Verify that the value is an integer and return it or convert the value to an integer and return it

Raises `TypeError` if unable to convert to an int

Parameters

`value` – String or int to convert of verify

Returns

int of given value

`ansible.module_utils.common.validation.`**`check_type_jsonarg`**`(`*value*`)`

Return a jsonified string. Sometimes the controller turns a json string into a dict/list so transform it back into json here

Raises `TypeError` if unable to covert the value

`ansible.module_utils.common.validation.`**`check_type_list`**`(`*value*`)`

Verify that the value is a list or convert to a list

A comma separated string will be split into a list. Raises a `TypeError` if unable to convert to a list.

Parameters

`value` – Value to validate or convert to a list

Returns

Original value if it is already a list, single item list if a float, int, or string without commas, or a multi-item list if a comma-delimited string.

`ansible.module_utils.common.validation.`**`check_type_path`**`(`*value*`)`

Verify the provided value is a string or convert it to a string, then return the expanded path

`ansible.module_utils.common.validation.`**`check_type_raw`**`(`*value*`)`

Returns the raw value

`ansible.module_utils.common.validation.`**`check_type_str`**`(`*value*, *allow_conversion=True*, *param=None*, *prefix=''*`)`

Verify that the value is a string or convert to a string.

Since unexpected changes can sometimes happen when converting to a string, `allow_conversion` controls whether or not the value will be converted or a TypeError will be raised if the value is not a string and would be converted

Parameters

- **`value`** – Value to validate or convert to a string
- **`allow_conversion`** – Whether to convert the string and return it or raise a TypeError

Returns

Original value if it is a string, the value converted to a string if allow_conversion=True, or raises a TypeError if allow_conversion=False.

`ansible.module_utils.common.validation.`**`count_terms`**`(`*terms*, *parameters*`)`

Count the number of occurrences of a key in a given dictionary

Parameters

- **`terms`** – String or iterable of values to check
- **`parameters`** – Dictionary of parameters

Returns

An integer that is the number of occurrences of the terms values in the provided dictionary.

1.38.4 Errors

`exception` `ansible.module_utils.errors.`**`AnsibleFallbackNotFound`**

Fallback validator was not found

`exception` `ansible.module_utils.errors.`**`AnsibleValidationError`**`(`*message*`)`

Single argument spec validation error

`error_message`

The error message passed in when the exception was raised.

`property msg`

The error message passed in when the exception was raised.

`exception` `ansible.module_utils.errors.`**`AnsibleValidationErrorMultiple`**`(`*errors=None*`)`

Multiple argument spec validation errors

`errors`

`list` of *`AnsibleValidationError`* objects

`property msg`

The first message from the first error in `errors`.

property messages

`list` of each error message in `errors`.

append(*error*)

Append a new error to `self.errors`.

Only *AnsibleValidationError* should be added.

extend(*errors*)

Append each item in `errors` to `self.errors`. Only *AnsibleValidationError* should be added.

exception `ansible.module_utils.errors.`**AliasError**(*message*)

Error handling aliases

exception `ansible.module_utils.errors.`**ArgumentTypeError**(*message*)

Error with parameter type

exception `ansible.module_utils.errors.`**ArgumentValueError**(*message*)

Error with parameter value

exception `ansible.module_utils.errors.`**DeprecationError**(*message*)

Error processing parameter deprecations

exception `ansible.module_utils.errors.`**ElementError**(*message*)

Error when validating elements

exception `ansible.module_utils.errors.`**MutuallyExclusiveError**(*message*)

Mutually exclusive parameters were supplied

exception `ansible.module_utils.errors.`**NoLogError**(*message*)

Error converting no_log values

exception `ansible.module_utils.errors.`**RequiredByError**(*message*)

Error with parameters that are required by other parameters

exception `ansible.module_utils.errors.`**RequiredDefaultError**(*message*)

A required parameter was assigned a default value

exception `ansible.module_utils.errors.`**RequiredError**(*message*)

Missing a required parameter

exception `ansible.module_utils.errors.`**RequiredIfError**(*message*)

Error with conditionally required parameters

exception `ansible.module_utils.errors.`**RequiredOneOfError**(*message*)

Error with parameters where at least one is required

exception `ansible.module_utils.errors.`**RequiredTogetherError**(*message*)

Error with parameters that are required together

exception `ansible.module_utils.errors.`**SubParameterTypeError**(*message*)

Incorrect type for subparameter

exception `ansible.module_utils.errors.`**UnsupportedError**(*message*)

Unsupported parameters were supplied

1.39 Special Variables

1.39.1 Magic variables

These variables cannot be set directly by the user; Ansible will always override them to reflect internal state.

ansible_check_mode
Boolean that indicates if we are in check mode or not

ansible_config_file
The full path of used Ansible configuration file

ansible_dependent_role_names
The names of the roles currently imported into the current play as dependencies of other plays

ansible_diff_mode
Boolean that indicates if we are in diff mode or not

ansible_forks
Integer reflecting the number of maximum forks available to this run

ansible_inventory_sources
List of sources used as inventory

ansible_limit
Contents of the `--limit` CLI option for the current execution of Ansible

ansible_loop
A dictionary/map containing extended loop information when enabled through `loop_control.extended`

ansible_loop_var
The name of the value provided to `loop_control.loop_var`. Added in `2.8`

ansible_index_var
The name of the value provided to `loop_control.index_var`. Added in `2.9`

ansible_parent_role_names
When the current role is being executed by means of an include_role or import_role action, this variable contains a list of all parent roles, with the most recent role (in other words, the role that included/imported this role) being the first item in the list. When multiple inclusions occur, this list lists the *last* role (in other words, the role that included this role) as the *first* item in the list. It is also possible that a specific role exists more than once in this list.

For example: When role **A** includes role **B**, inside role B, `ansible_parent_role_names` will equal to `['A']`. If role **B** then includes role **C**, the list becomes `['B', 'A']`.

ansible_parent_role_paths
When the current role is being executed by means of an include_role or import_role action, this variable contains a list of all parent roles paths, with the most recent role (in other words, the role that included/imported this role) being the first item in the list. Please refer to `ansible_parent_role_names` for the order of items in this list.

ansible_play_batch
List of active hosts in the current play run limited by the serial, aka 'batch'. Failed/Unreachable hosts are not considered 'active'.

ansible_play_hosts
List of hosts in the current play run, not limited by the serial. Failed/Unreachable hosts are excluded from this list.

ansible_play_hosts_all
List of all the hosts that were targeted by the play

ansible_play_role_names
The names of the roles currently imported into the current play. This list does **not** contain the role names that are implicitly included through dependencies.

ansible_playbook_python
The path to the python interpreter being used by Ansible on the controller

ansible_role_names
The names of the roles currently imported into the current play, or roles referenced as dependencies of the roles imported into the current play.

ansible_role_name
The fully qualified collection role name, in the format of `namespace.collection.role_name`

ansible_collection_name
The name of the collection the task that is executing is a part of. In the format of `namespace.collection`

ansible_run_tags
Contents of the `--tags` CLI option, which specifies which tags will be included for the current run. Note that if `--tags` is not passed, this variable will default to `["all"]`.

ansible_search_path
Current search path for action plugins and lookups, in other words, where we search for relative paths when you do `template: src=myfile`

ansible_skip_tags
Contents of the `--skip-tags` CLI option, which specifies which tags will be skipped for the current run.

ansible_verbosity
Current verbosity setting for Ansible

ansible_version
Dictionary/map that contains information about the current running version of ansible, it has the following keys: full, major, minor, revision and string.

group_names
List of groups the current host is part of

groups
A dictionary/map with all the groups in inventory and each group has the list of hosts that belong to it

hostvars
A dictionary/map with all the hosts in inventory and variables assigned to them

inventory_hostname
The inventory name for the 'current' host being iterated over in the play

inventory_hostname_short
The short version of *inventory_hostname*

inventory_dir
The directory of the inventory source in which the *inventory_hostname* was first defined

inventory_file
The file name of the inventory source in which the *inventory_hostname* was first defined

omit
Special variable that allows you to 'omit' an option in a task, for example - `user: name=bob home={{ bobs_home|default(omit) }}`

play_hosts
Deprecated, the same as ansible_play_batch

ansible_play_name
: The name of the currently executed play. Added in `2.8`. (*name* attribute of the play, not file name of the playbook.)

playbook_dir
: The path to the directory of the current playbook being executed. NOTE: This might be different than directory of the playbook passed to the `ansible-playbook` command line when a playbook contains a `import_playbook` statement.

role_name
: The name of the role currently being executed.

role_names
: Deprecated, the same as ansible_play_role_names

role_path
: The path to the dir of the currently running role

1.39.2 Facts

These are variables that contain information pertinent to the current host (*inventory_hostname*). They are only available if gathered first. See *Discovering variables: facts and magic variables* for more information.

ansible_facts
: Contains any facts gathered or cached for the *inventory_hostname* Facts are normally gathered by the setup module automatically in a play, but any module can return facts.

ansible_local
: Contains any 'local facts' gathered or cached for the *inventory_hostname*. The keys available depend on the custom facts created. See the setup module and *facts.d or local facts* for more details.

1.39.3 Connection variables

Connection variables are normally used to set the specifics on how to execute actions on a target. Most of them correspond to connection plugins, but not all are specific to them; other plugins like shell, terminal and become are normally involved. Only the common ones are described as each connection/become/shell/etc plugin can define its own overrides and specific variables. See *Controlling how Ansible behaves: precedence rules* for how connection variables interact with *configuration settings*, *command-line options*, and *playbook keywords*.

ansible_become_user
: The user Ansible 'becomes' after using privilege escalation. This must be available to the 'login user'.

ansible_connection
: The connection plugin actually used for the task on the target host.

ansible_host
: The ip/name of the target host to use instead of *inventory_hostname*.

ansible_python_interpreter
: The path to the Python executable Ansible should use on the target host.

ansible_user
: The user Ansible 'logs in' as.

1.40 Red Hat Ansible Automation Platform

Important: Red Hat Ansible Automation Platform is available on multiple cloud platforms. See Ansible on Clouds for details.

Red Hat Ansible Automation Platform (RHAAP) is an integrated solution for operationalizing Ansible across your team, organization, and enterprise. The platform includes a controller with a web console and REST API, analytics, execution environments, and much more.

RHAAP gives you role-based access control, including control over the use of securely stored credentials for SSH and other services. You can sync your inventory with a wide variety of cloud sources, and powerful multi-playbook workflows allow you to model complex processes.

RHAAP logs all of your jobs, integrates well with LDAP, SAML, and other authentication sources, and has an amazing browsable REST API. Command line tools are available for easy integration with Jenkins as well.

RHAAP incorporates the downstream Red Hat supported product version of Ansible AWX, the downstream Red Hat supported product version of Ansible Galaxy, and multiple SaaS offerings. Find out more about RHAAP features and on the Red Hat Ansible Automation Platform webpage. A Red Hat Ansible Automation Platform subscription includes support from Red Hat, Inc.

1.41 Ansible Automation Hub

Important: Red Hat Ansible Automation Platform will soon be available on Microsoft Azure. Sign up to preview the experience.

Ansible Automation Hub is the official location to discover and download supported *collections*, included as part of an Ansible Automation Platform subscription. These content collections contain modules, plugins, roles, and playbooks in a downloadable package.

Ansible Automation Hub gives you direct access to trusted content collections from Red Hat and Certified Partners. You can find content by topic or Ansible Partner organizations.

Ansible Automation Hub is the downstream Red Hat supported product version of Ansible Galaxy. Find out more about Ansible Automation Hub features and how to access it at Ansible Automation Hub. Ansible Automation Hub is part of the Red Hat Ansible Automation Platform subscription, and comes bundled with support from Red Hat, Inc.

1.42 Logging Ansible output

By default Ansible sends output about plays, tasks, and module arguments to your screen (STDOUT) on the control node. If you want to capture Ansible output in a log, you have three options:

- To save Ansible output in a single log on the control node, set the `log_path` *configuration file setting*. You may also want to set `display_args_to_stdout`, which helps to differentiate similar tasks by including variable values in the Ansible output.
- To save Ansible output in separate logs, one on each managed node, set the `no_target_syslog` and `syslog_facility` *configuration file settings*.
- To save Ansible output to a secure database, use AWX or *Red Hat Ansible Automation Platform*. You can then review history based on hosts, projects, and particular inventories over time, using graphs and/or a REST API.

1.42.1 Protecting sensitive data with `no_log`

If you save Ansible output to a log, you expose any secret data in your Ansible output, such as passwords and user names. To keep sensitive values out of your logs, mark tasks that expose them with the `no_log: True` attribute. However, the `no_log` attribute does not affect debugging output, so be careful not to debug playbooks in a production environment. See *How do I keep secret data in my playbook?* for an example.

1.43 Ansible Roadmap

The Ansible team develops a roadmap for each major and minor Ansible release. The latest roadmap shows current work; older roadmaps provide a history of the project. We don't publish roadmaps for subminor versions. So 2.10 and 2.11 have roadmaps, but 2.10.1 does not.

We incorporate team and community feedback in each roadmap, and aim for further transparency and better inclusion of both community desires and submissions.

Each roadmap offers a *best guess*, based on the Ansible team's experience and on requests and feedback from the community, of what will be included in a given release. However, some items on the roadmap may be dropped due to time constraints, lack of community maintainers, and so on.

Each roadmap is published both as an idea of what is upcoming in Ansible, and as a medium for seeking further feedback from the community.

You can submit feedback on the current roadmap in multiple ways:

- Edit the agenda of an Ansible Community Meeting (preferred)
- Post on the `#ansible-community` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat)

See *Ansible communication channels* for details on how to join and use our chat channels.

1.43.1 Ansible project 8.0

This release schedule includes dates for the ansible package, with a few dates for the ansible-core package as well. All dates are subject to change. See the ansible-core 2.15 Roadmap for the most recent updates on `ansible-core`.

Release schedule

2023-03-31
ansible-core feature freeze, stable-2.15 branch created.

2023-04-03
Start of ansible-core 2.15 betas (biweekly, as needed).

2023-04-04
Ansible-8.0.0 alpha1 (roughly biweekly `ansible` alphas timed to coincide with `ansible-core-2.15` pre-releases).

2023-04-24
First ansible-core 2.15 release candidate.

2023-05-02
Another Ansible-8.0.0 alpha release.

2023-05-15
Ansible-core-2.15.0 released.

2023-05-15
Last day for collections to make backwards incompatible releases that will be accepted into Ansible-8. This includes adding new collections to Ansible 8.0.0; from now on new collections have to wait for 8.1.0 or later.

2023-05-16
Ansible-8.0.0 beta1 – feature freeze[1] (weekly beta releases; collection owners and interested users should test for bugs).

2023-05-23
Ansible-8.0.0 rc1[2][3] (weekly release candidates as needed; test and alert us to any blocker bugs). Blocker bugs will slip release.

2023-05-26
Last day to trigger an Ansible-8.0.0rc2 release because of major defects in Ansible-8.0.0rc1.

2023-05-30
Ansible-8.0.0rc2 when necessary, otherwise Ansible-8.0.0 release.

2023-06-06
Ansible-8.0.0 release when Ansible-8.0.0rc2 was necessary.

2023-05-30 or 2023-06-06
Create the ansible-build-data directory and files for Ansible-9.

2023-06-12
Release of ansible-core 2.15.1.

2023-06-13
Release of Ansible-8.1.0 (bugfix + compatible features: every four weeks.)

Note: Breaking changes will be introduced in Ansible 8.0.0, although we encourage the use of deprecation periods that will show up in at least one Ansible release before the breaking change happens, this is not guaranteed.

Ansible minor releases

Ansible 8.x minor releases will occur approximately every four weeks if changes to collections have been made or to align to a later ansible-core-2.15.x. Ansible 8.x minor releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed. For example, if Ansible-8.0.0 ships with community.crypto 2.3.0; Ansible-8.1.0 may ship with community.crypto 2.4.0 but would not ship with community.crypto 3.0.0.

[1] No new modules or major features accepted after this date. In practice, this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community.crypto 2.3.0; Ansible-8.0.0 could ship with community.crypto 2.3.1. It would not ship with community.crypto 2.4.0.

[2] After this date only changes blocking a release are accepted. Accepted changes require creating a new rc and may slip the final release date.

[3] Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at a community IRC meeting (before this freeze) to decide whether to bump the version of the collection for a fix. See the Community IRC meeting agenda.

Note: Minor releases will stop when Ansible-8 is released. See the *Release and Maintenance Page* for more information.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

1.43.2 Ansible project 7.0

This release schedule includes dates for the ansible package, with a few dates for the ansible-core package as well. All dates are subject to change. See the ansible-core 2.14 Roadmap for the most recent updates on `ansible-core`.

Release schedule

2022-09-19
ansible-core feature freeze, stable-2.14 branch created.

2022-09-26
Start of ansible-core 2.14 betas (biweekly, as needed).

2022-09-27
Ansible-7.0.0 alpha1 (roughly biweekly `ansible` alphas timed to coincide with `ansible-core-2.14` pre-releases).

2022-10-12
Community topic: List any backwards incompatible collection releases that beta1 should try to accommodate.

2022-10-17
First ansible-core 2.14 release candidate.

2022-10-25
Ansible-7.0.0 alpha2.

2022-10-26
Community Meeting topic: Decide what contingencies to activate for any blockers that do not meet the deadline.

2022-11-07
Ansible-core-2.14.0 released.

2022-11-07
Last day for collections to make backwards incompatible releases that will be accepted into Ansible-7. This includes adding new collections to Ansible 7.0.0; from now on new collections have to wait for 7.1.0 or later.

2022-11-08
Create the ansible-build-data directory and files for Ansible-8.

2022-11-08
Ansible-7.0.0 beta1 – feature freeze[1] (weekly beta releases; collection owners and interested users should test for bugs).

2022-11-15
Ansible-7.0.0 rc1[2][3] (weekly release candidates as needed; test and alert us to any blocker bugs). Blocker bugs will slip release.

2022-11-18
Last day to trigger an Ansible-7.0.0rc2 release because of major defects in Ansible-7.0.0rc1.

2022-11-22
Ansible-7.0.0rc2 when necessary, otherwise Ansible-7.0.0 release.

2022-11-29
Ansible-7.0.0 release when Ansible-7.0.0rc2 was necessary.

2022-12-05
Release of ansible-core 2.14.1.

2022-12-06
Release of Ansible-7.1.0 (bugfix + compatible features: every four weeks.)

Note: Breaking changes will be introduced in Ansible 7.0.0, although we encourage the use of deprecation periods that will show up in at least one Ansible release before the breaking change happens, this is not guaranteed.

Ansible minor releases

Ansible 7.x minor releases will occur approximately every four weeks if changes to collections have been made or to align to a later ansible-core-2.14.x. Ansible 7.x minor releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed. For example, if Ansible-7.0.0 ships with community.crypto 2.3.0; Ansible-7.1.0 may ship with community.crypto 2.4.0 but would not ship with community.crypto 3.0.0.

Note: Minor releases will stop when Ansible-8 is released. See the *Release and Maintenance Page* for more information.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

[1] No new modules or major features accepted after this date. In practice, this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community.crypto 2.3.0; Ansible-7.0.0 could ship with community.crypto 2.3.1. It would not ship with community.crypto 2.4.0.

[2] After this date only changes blocking a release are accepted. Accepted changes require creating a new rc and may slip the final release date.

[3] Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at a community IRC meeting (before this freeze) to decide whether to bump the version of the collection for a fix. See the Community IRC meeting agenda.

1.43.3 Ansible project 6.0

This release schedule includes dates for the ansible package, with a few dates for the ansible-core package as well. All dates are subject to change. See the ansible-core 2.13 Roadmap for the most recent updates on `ansible-core`.

Release schedule

2022-03-28
ansible-core feature freeze, stable-2.13 branch created.

2022-04-11
Start of ansible-core 2.13 betas (biweekly, as needed).

2022-04-12
Ansible-6.0.0 alpha1 (roughly biweekly `ansible` alphas timed to coincide with `ansible-core-2.13` pre-releases).

2022-04-27
Community Meeting topic: List any backwards incompatible collection releases that beta1 should try to accommodate.

2022-05-02
First ansible-core release candidate.

2022-05-03
Ansible-6.0.0 alpha2.

2022-05-11
Community Meeting topic: Decide what contingencies to activate for any blockers that do not meet the deadline.

2022-05-16
Ansible-core-2.13 released.

2022-05-17
Ansible-6.0.0 alpha3.

2022-05-23
Last day for collections to make backwards incompatible releases that will be accepted into Ansible-6. This includes adding new collections to Ansible 6.0.0; from now on new collections have to wait for 6.1.0 or later.

2022-05-24
Create the ansible-build-data directory and files for Ansible-7.

2022-05-24
Ansible-6.0.0 beta1 – feature freeze[1] (weekly beta releases; collection owners and interested users should test for bugs).

[1] No new modules or major features accepted after this date. In practice, this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community.crypto 2.3.0; Ansible-6.0.0 could ship with community.crypto 2.3.1. It would not ship with community.crypto 2.4.0.

2022-05-31
Ansible-6.0.0 beta2.

2022-06-07
Ansible-6.0.0 rc1[2][3] (weekly release candidates as needed; test and alert us to any blocker bugs). Blocker bugs will slip release.

2022-06-21
Ansible-6.0.0 release.

2022-07-12
Release of Ansible-6.1.0 (bugfix + compatible features: every three weeks.)

Note: Breaking changes will be introduced in Ansible 6.0.0, although we encourage the use of deprecation periods that will show up in at least one Ansible release before the breaking change happens, this is not guaranteed.

Ansible minor releases

Ansible 6.x minor releases will occur approximately every three weeks if changes to collections have been made or if it is deemed necessary to force an upgrade to a later ansible-core-2.13.x. Ansible 6.x minor releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed. For example, if Ansible-6.0.0 ships with community.crypto 2.3.0; Ansible-6.1.0 may ship with community.crypto 2.4.0 but would not ship with community.crypto 3.0.0.

Note: Minor releases will stop when Ansible-7 is released. See the *Release and Maintenance Page* for more information.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

Planned work

More details can be found in the community-topics planning issue.

- Remove compatibility code which prevents parallel install of Ansible 6 with Ansible 2.9 or ansible-base 2.10
- Stop installing files (such as tests and development artifacts like editor configs) we have no use for
- Ship Python wheels (as ansible-core 2.13 will likely also do) to improve installation performance

[2] After this date only changes blocking a release are accepted. Accepted changes require creating a new rc and may slip the final release date.
[3] Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at a community IRC meeting (before this freeze) to decide whether to bump the version of the collection for a fix. See the Community IRC meeting agenda.

1.43.4 Ansible project 5.0

This release schedule includes dates for the ansible package, with a few dates for the ansible-core package as well. All dates are subject to change. See *Ansible-core 2.12* for the most recent updates on `ansible-core`.

Release schedule

2021-04-14
New Collections can be reviewed for inclusion in Ansible 5. Submit a request to include a new collection in this GitHub Discussion.

2021-09-24
ansible-core feature freeze.

2021-09-27
Start of ansible-core 2.12 betas (weekly, as needed).

2021-10-05
Ansible-5.0.0 alpha1 (roughly biweekly `ansible` alphas timed to coincide with `ansible-core-2.12` pre-releases).

2021-10-12
Last day for new collections to be submitted for inclusion in Ansible-5. Collections MUST be reviewed and approved before being included. There is no guarantee that we will review every collection. The earlier your collection is submitted, the more likely it will be that your collection will be reviewed and the necessary feedback can be addressed in time for inclusion.

2021-10-13
Community Meeting topic: List any new collection reviews which block release. List any backwards incompatible collection releases that beta1 should try to accommodate.

2021-10-18
First ansible-core release candidate, stable-2.12 branch created.

2021-10-19
Ansible-5.0.0 alpha2.

2021-10-26
Last day for new collections to be **reviewed and approved** for inclusion in Ansible-5.

2021-10-27
Community Meeting topic: Decide what contingencies to activate for any blockers that do not meet the deadline.

2021-11-02
Ansible-5.0.0 alpha3.

2021-11-08
Ansible-core-2.12 released.

2021-11-08
Last day for collections to make backwards incompatible releases that will be accepted into Ansible-5.

2021-11-09
Create the ansible-build-data directory and files for Ansible-6. New collection approvals will target this.

2021-11-09
Ansible-5.0.0 beta1 – feature freeze[1] (weekly beta releases; collection owners and interested users should test for bugs).

2021-11-16
Ansible-5.0.0 beta2.

2021-11-23
Ansible-5.0.0 rc1[2][3] (weekly release candidates as needed; test and alert us to any blocker bugs). Blocker bugs will slip release.

2021-11-30
Ansible-5.0.0 release.

2021-12-21
Release of Ansible-5.1.0 (bugfix + compatible features: every three weeks.)

Note: Breaking changes will be introduced in Ansible 5.0.0, although we encourage the use of deprecation periods that will show up in at least one Ansible release before the breaking change happens, this is not guaranteed.

Ansible minor releases

Ansible 5.x minor releases will occur approximately every three weeks if changes to collections have been made or if it is deemed necessary to force an upgrade to a later ansible-core-2.12.x. Ansible 5.x minor releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed. For example, if Ansible-5.0.0 ships with community.crypto 2.1.0; Ansible-5.1.0 may ship with community.crypto 2.2.0 but would not ship with community.crypto 3.0.0.

Note: Minor releases will stop when Ansible-6 is released. See the *Release and Maintenance Page* for more information.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

[1] No new modules or major features accepted after this date. In practice, this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community.crypto 2.1.0; Ansible-5.0.0 could ship with community.crypto 2.1.1. It would not ship with community.crypto 2.2.0.

[2] After this date only changes blocking a release are accepted. Accepted changes require creating a new rc and may slip the final release date.

[3] Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at a community IRC meeting (before this freeze) to decide whether to bump the version of the collection for a fix. See the Community IRC meeting agenda.

1.43.5 Ansible project 4.0

This release schedule includes dates for the ansible package, with a few dates for the ansible-core package as well. All dates are subject to change. See *Ansible-core 2.11* for the most recent updates on `ansible-core`.

Release schedule

2021-01-26
New Collections will be reviewed for inclusion in Ansible 4. Submit a request to include a new collection in this GitHub Discussion.

2021-03-03
Ansible-4.0.0 alpha1 (biweekly `ansible` alphas. These are timed to coincide with the start of the `ansible-core-2.11` pre-releases).

2021-03-16
Ansible-4.0.0 alpha2

2021-03-30
Ansible-4.0.0 alpha3

2021-04-13
Last day for new collections to be submitted for inclusion in Ansible 4. Note that collections MUST be reviewed and approved before being included. There is no guarantee that we will review every collection. The earlier your collection is submitted, the more likely it will be that your collection will be reviewed and the necessary feedback can be addressed in time for inclusion.

2021-04-13
Ansible-4.0.0 alpha4

2021-04-14
Community Meeting topic: list any new collection reviews which block release. List any backwards incompatible collection releases that beta1 should try to accommodate.

2021-04-21
Community Meeting topic: Decide what contingencies to activate for any blockers that do not meet the deadline.

2021-04-26
Last day for new collections to be **reviewed and approved** for inclusion in Ansible 4.

2021-04-26
Last day for collections to make backwards incompatible releases that will be accepted into Ansible 4.

2021-04-27
Ansible-4.0.0 beta1 – feature freeze[1] (weekly beta releases. Collection owners and interested users should test for bugs).

2021-05-04
Ansible-4.0.0 beta2

[1] No new modules or major features accepted after this date. In practice, this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community-crypto-2.1.0; ansible-3.0.0 could ship with community-crypto-2.1.1. It would not ship with community-crypto-2.2.0.

2021-05-11
Ansible-4.0.0 rc1[2][3] (weekly release candidates as needed. Test and alert us to any blocker bugs).

2021-05-18
Ansible-4.0.0 release

2021-06-08
Release of Ansible-4.1.0 (bugfix + compatible features: every three weeks)

Note: Breaking changes will be introduced in Ansible 4.0.0, although we encourage the use of deprecation periods that will show up in at least one Ansible release before the breaking change happens, this is not guaranteed.

Ansible minor releases

Ansible 4.x minor releases will occur approximately every three weeks if changes to collections have been made or if it is deemed necessary to force an upgrade to a later ansible-core-2.11.x. Ansible 4.x minor releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed. For example, if Ansible-4.0.0 ships with community-crypto-2.1.0; Ansible-4.1.0 may ship with community-crypto-2.2.0 but would not ship with community-crypto-3.0.0).

Note: Minor releases will stop when Ansible-5 is released. See the *Release and Maintenance Page* for more information.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

1.43.6 Ansible project 3.0

This release schedule includes dates for the ansible package, with a few dates for the ansible-base package as well. All dates are subject to change. Ansible 3.x.x includes `ansible-base` 2.10. See *Ansible-base 2.10* for the most recent updates on `ansible-base`.

[2] After this date only changes blocking a release are accepted. Accepted changes require creating a new rc and may slip the final release date.

[3] Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at a Community meeting (before this freeze) to decide whether to bump the version of the collection for a fix. See the Community meeting agenda.

Release schedule

Note: Ansible is switching from its traditional versioning scheme to semantic versioning starting with this release. So this version is 3.0.0 instead of 2.11.0.

2020-12-16
Finalize rules for net-new collections submitted for the ansible release.

2021-01-27
Final day for new collections to be **reviewed and approved**. They MUST be submitted prior to this to give reviewers a chance to look them over and for collection owners to fix any problems.

2021-02-02
Ansible-3.0.0-beta1 – feature freeze[1]

2021-02-09
Ansible-3.0.0-rc1 – final freeze[2][3]

2021-02-16
Release of Ansible-3.0.0

2021-03-09
Release of Ansible-3.1.0 (bugfix + compatible features: every three weeks)

Note: Breaking changes may be introduced in Ansible 3.0.0, although we encourage collection owners to use deprecation periods that will show up in at least one Ansible release before the breaking change happens.

Ansible minor releases

Ansible 3.x.x minor releases will occur approximately every three weeks if changes to collections have been made or if it is deemed necessary to force an upgrade to a later ansible-base-2.10.x. Ansible 3.x.x minor releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed. For example, if Ansible-3.0.0 ships with community-crypto-2.1.0; Ansible-3.1.0 may ship with community-crypto-2.2.0 but would not ship with community-crypto-3.0.0).

Note: Minor releases will stop when *Ansible-4* is released. See the *Release and Maintenance Page* for more information.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

[1] No new modules or major features accepted after this date. In practice this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community-crypto-2.1.0; ansible-3.0.0 could ship with community-crypto-2.1.1. It would not ship with community-crypto-2.2.0.

[2] After this date only changes blocking a release are accepted. Accepted changes require creating a new rc and may slip the final release date.

[3] Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at a community meeting (before this freeze) to decide whether to bump the version of the collection for a fix. See the Community meeting agenda.

ansible-base release

Ansible 3.x.x works with `ansible-base` 2.10. See *Ansible-base 2.10* for details.

1.43.7 Ansible project 2.10

This release schedule includes dates for the ansible package, with a few dates for the ansible-base package as well. All dates are subject to change. See *Ansible-base 2.10* for the most recent updates on ansible-base.

Release Schedule

Note: Dates subject to change.

Note: We plan to post weekly alpha releases to the PyPI ansible project for testing.

Warning: We initially were going to have feature freeze on 2020-08-18. We tried this but decided to change course. Instead, we'll enter feature freeze when ansible-2.10.0 beta1 is released.

- 2020-06-23: ansible-2.10 alpha freeze. No net new collections will be added to the `ansible-2.10` package after this date.
- 2020-07-10: Ansible collections freeze date for content shuffling. Content should be in its final collection for the ansible-2.10 series of releases. No more content should move out of the `community.general` or `community.network` collections.
- 2020-08-13: ansible-base 2.10 Release date, see *Ansible-base 2.10*.
- 2020-08-14: final ansible-2.10 alpha.
- 2020-09-01: ansible-2.10.0 beta1 and feature freeze.
  - No new modules or major features will be added after this date. In practice this means we will freeze the semver collection versions to compatible release versions. For example, if the version of community.crypto on this date was community-crypto-1.1.0; ansible-2.10.0 could ship with community-crypto-1.1.1. It would not ship with community-crypto-1.2.0.
- 2020-09-08: ansible-2.10.0 beta2.
- 2020-09-15: ansible-2.10.0 rc1 and final freeze.
  - After this date only changes blocking a release are accepted.
  - Collections will only be updated to a new version if a blocker is approved. Collection owners should discuss any blockers at the community meeting (on 9-17) to decide whether to bump the version of the collection for a fix. See the Community meeting agenda.

** Additional release candidates to be published as needed as blockers are fixed **

- 2020-09-22: ansible-2.10 GA release date.

Ansible-2.10.x patch releases will occur roughly every three weeks if changes to collections have been made or if it is deemed necessary to force an upgrade to a later ansible-base-2.10.x. Ansible-2.10.x patch releases may contain new features but not backwards incompatibilities. In practice, this means we will include new collection versions where either the patch or the minor version number has changed but not when the major number has changed (example: Ansible-2.10 ships with community-crypto-1.1.0; ansible-2.10.1 may ship with community-crypto-1.2.0 but would not ship with community-crypto-2.0.0).

Note: Minor releases will stop when *Ansible-3* is released. See the *Release and Maintenance Page* for more information.

Breaking changes may be introduced in ansible-3.0 although we encourage collection owners to use deprecation periods that will show up in at least one Ansible release before being changed incompatibly.

For more information, reach out on a mailing list or a chat channel - see *Communicating with the Ansible community* for more details.

1.43.8 Older Roadmaps

Older roadmaps are listed here to provide a history of the Ansible project.

See *Ansible Roadmap* to find current Ansible and `ansible-base` roadmaps.

Ansible 2.9

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this Ansible release.

Note: There is no Alpha phase in 2.9.

- 2019-08-29 Beta 1 **Feature freeze** No new functionality (including modules/plugins) to any code
- 2019-09-19 Release Candidate 1
- 2019-10-03 Release Candidate 2
- 2019-10-10 Release Candidate 3
- 2019-10-17 Release Candidate 4 (if needed)
- 2019-10-24 Release Candidate 5 (if needed)
- 2019-10-31 Release

Release Manager

TBD

Temporarily, Matt Davis (@nitzmahone) or Matt Clay (@mattclay) on IRC or github.

Planned work

See the Ansible 2.9 Project Board

Ansible 2.8

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this Ansible release.

- 2019-04-04 Alpha 1 **Core freeze** No new features to `support:core` code. Includes no new options to existing Core modules
- 2019-04-11 Beta 1 **Feature freeze** No new functionality (including modules/plugins) to any code
- 2019-04-25 Release Candidate 1
- 2019-05-02 Release Candidate 2
- 2019-05-10 Release Candidate 3
- 2019-05-16 Release

Release Manager

Toshio Kuratomi (IRC: abadger1999; GitHub: @abadger)

Planned work

See the Ansible 2.8 Project Board

Ansible 2.7

Topics

Release Schedule

Expected

- 2018-08-23 Core Freeze (Engine and Core Modules/Plugins)
- 2018-08-23 Alpha Release 1
- 2018-08-30 Community Freeze (Non-Core Modules/Plugins)
- 2018-08-30 Beta Release 1
- 2018-09-06 Release Candidate 1 (If needed)
- 2018-09-13 Release Candidate 2 (If needed)
- 2018-09-20 Release Candidate 3 (If needed)
- 2018-09-27 Release Candidate 4 (If needed)

- 2018-10-04 General Availability

Release Manager

Toshio Kuratomi (IRC: abadger1999; GitHub: @abadger)

Cleaning Duty

- Drop Py2.6 for controllers Docs PR #42971 and issue #42972
- Remove dependency on simplejson issue #42761

Engine Improvements

- Performance improvement invoking Python modules pr #41749
- Jinja native types will allow for users to render a Python native type. pr #32738

Core Modules

- Include feature changes and improvements
  - Create new argument `apply` that will allow for included tasks to inherit explicitly provided attributes. pr #39236
  - Create "private" functionality for allowing vars/default to be exposed outside of roles. pr #41330
- Provide a parameter for the `template` module to output to different encoding formats pr #42171
- `reboot` module for Linux hosts (@samdoran) pr #35205

Cloud Modules

General

- Cloud auth plugin proposal #24

AWS

- Inventory plugin for RDS pr #41919
- Count support for *ec2_instance*
- *aws_eks* module pr #41183
- Cloudformation stack sets support (PR#41669)
- RDS instance and snapshot modules pr #39994 pr #43789
- Diff mode improvements for cloud modules pr #44533

Azure

- Azure inventory plugin issue #42769

Network

General

- Refactor the APIs in cliconf (issue #39056) and netconf (issue #39160) plugins so that they have a uniform signature across supported network platforms. **done** (PR #41846) (PR #43643) (PR #43837) (PR #43203) (PR #42300) (PR #44157)

Modules

- New `cli_config` module issue #39228 **done** PR #42413.
- New `cli_command` module issue #39284
- Refactor `netconf_config` module to add additional functionality. **done** proposal #104 (PR #44379)

Windows

General

- Added new connection plugin that uses PSRP as the connection protocol pr #41729

Modules

- Revamp Chocolatey to fix bugs and support offline installation pr #43013.
- Add Chocolatey modules that can manage the following Chocolatey features
  - Sources pr #42790
  - Features pr #42848
  - Config pr #42915

Ansible 2.6

Topics

Release Schedule

Actual

- 2018-05-17 Core Freeze (Engine and Core Modules/Plugins)
- 2018-05-21 Alpha Release 1
- 2018-05-25 Community Freeze (Non-Core Modules/Plugins)
- 2018-05-25 Branch stable-2.6
- 2018-05-30 Alpha Release 2
- 2018-06-05 Release Candidate 1
- 2018-06-08 Release Candidate 2
- 2018-06-18 Release Candidate 3
- 2018-06-25 Release Candidate 4
- 2018-06-26 Release Candidate 5
- 2018-06-28 Final Release

Release Manager

- 2.6.0-2.6.12 Matt Clay (IRC/GitHub: @mattclay)
- 2.6.13+ Toshio Kuratomi (IRC: abadger1999; GitHub: @abadger)

Engine improvements

- Version 2.6 is largely going to be a stabilization release for Core code.
- Some of the items covered in this release, but are not limited to are the following:
  - `ansible-inventory`
  - `import_*`
  - `include_*`
  - Test coverage
  - Performance Testing

Core Modules

- Adopt-a-module Campaign
  - Review current status of all Core Modules
  - Reduce backlog of open issues against these modules

Cloud Modules

Network

Connection work

- New connection plugin: eAPI proposal#102
- New connection plugin: NX-API
- Support for configurable options for network_cli & netconf

Modules

- New `net_get` - platform independent module for pulling configuration via SCP/SFTP over network_cli
- New `net_put` - platform independent module for pushing configuration via SCP/SFTP over network_cli
- New `netconf_get` - Netconf module to fetch configuration and state data proposal#104

Other Features

- Stretch & tech preview: Configuration caching for network_cli. Opt-in feature to avoid `show running` performance hit

Windows

Ansible 2.5

Core Engine Freeze and Module Freeze: 22 January 2018

Core and Curated Module Freeze: 22 January 2018

Community Module Freeze: 7 February 2018

Release Candidate 1 will be 21 February, 2018

Target: March 2018

Service Release schedule: every 2-3 weeks

Topics

Release Manager

Matt Davis (IRC/GitHub: @nitzmahone)

Engine improvements

- Assemble module improvements - assemble just skips when in check mode, it should be able to test if there is a difference and changed=true/false. - The same with diff, it should work as template modules does
- Handle Password reset prompts cleaner
- Tasks stats for rescues and ignores
- Normalize temp dir usage across all subsystems
- Add option to set playbook dir for adhoc, inventory and console to allow for 'relative path loading'

Ansible-Config

- Extend config to more plugin types and update plugins to support the new config

Inventory

- ansible-inventory option to output group variable assignment and data (–export)
- Create inventory plugins for: - aws

Facts

- Namespacing fact variables (via a config option) implemented in ansible/ansible PR #18445. Proposal found in ansible/proposals issue #17.
- Make fact collectors and gather_subset specs finer grained
- Eliminate unneeded deps between fact collectors
- Allow fact collectors to indicate if they need information from another fact collector to be gathered first.

Static Loop Keyword

- A simpler alternative to `with_`, `loop:` only takes a list
- Remove complexity from loops, lookups are still available to users
- Less confusing having a static directive vs a one that is dynamic depending on plugins loaded.

Vault

- Vault secrets client inc new 'keyring' client

Runtime Check on Modules for Disabling

- Filter on things like "supported_by" in module metadata
- Provide users with an option of "warning, error or allow/ignore"
- Configurable via ansible.cfg and environment variable

Windows

- Implement gather_subset on Windows facts
- Fix Windows async + become to allow them to work together
- Implement Windows become flags for controlling various modes **(done)** - logontype - elevation behavior
- Convert win_updates to action plugin for auto reboot and extra features **(done)**
- Spike out changing the connection over to PSRP instead of WSMV **(done- it's possible)**
- Module updates
  - win_updates **(done)**
    - Fix win_updates to detect (or request) become
    - Add enable/disable features to win_updates

  - win_dsc further improvements **(done)**

General Cloud

- Make multi-cloud provisioning easier
- Diff mode will output provisioning task results of ansible-playbook runs
- Terraform module

AWS

- Focus on pull requests for various modules
- Triage existing merges for modules
- Module work
  - ec2_instance
  - ec2_vpc: Allow the addition of secondary IPv4 CIDRS to existing VPCs.
  - AWS Network Load Balancer support (NLB module, ASG support, and so on)
  - rds_instance

Azure

- Azure CLI auth **(done)**
- Fix Azure module results to have "high-level" output instead of raw REST API dictionary **(partial, more to come in 2.6)**
- Deprecate Azure automatic storage accounts in azure_rm_virtualmachine **(breaks on Azure Stack, punted until AS supports managed disks)**

Network Roadmap

- Refactor common network shared code into package **(done)**
- Convert various nxos modules to use declarative intent **(done)**
- Refactor various modules to use the cliconf plugin **(done)**
- Add various missing declarative modules for supported platforms and functions **(done)**
- Implement a feature that handles platform differences and feature unavailability **(done)**
- netconf-config.py should provide control for deployment strategy
- Create netconf connection plugin **(done)**
- Create netconf fact module
- Turn network_cli into a usable connection type **(done)**
- Implements jsonrpc message passing for ansible-connection **(done)**
- Improve logging for ansible-connection **(done)**

- Improve stdout output for failures whilst using persistent connection **(done)**
- Create IOS-XR NetConf Plugin and refactor iosxr modules to use netconf plugin **(done)**
- Refactor junos modules to use netconf plugin **(done)**
- Filters: Add a filter to convert XML response from a network device to JSON object **(done)**

Documentation

- Extend documentation to more plugins
- Document vault-password-client scripts.
- Network Documentation
  - New landing page (to replace intro_networking) **(done)**
  - Platform specific guides **(done)**
  - Walk through: Getting Started **(done)**
  - Networking and `become` **(done)**
  - Best practice **(done)**

Contributor Quality of Life

- Finish PSScriptAnalyer integration with ansible-test (for enforcing Powershell style) **(done)**
- Resolve issues requiring skipping of some integration tests on Python 3.

1.44 ansible-core Roadmaps

The `ansible-core` team develops a roadmap for each major and minor `ansible-core` release. The latest roadmap shows current work; older roadmaps provide a history of the project. We don't publish roadmaps for subminor versions. So 2.10 and 2.11 have roadmaps, but 2.10.1 does not.

We incorporate team and community feedback in each roadmap, and aim for further transparency and better inclusion of both community desires and submissions.

Each roadmap offers a *best guess*, based on the `ansible-core` team's experience and on requests and feedback from the community, of what will be included in a given release. However, some items on the roadmap may be dropped due to time constraints, lack of community maintainers, and so on.

Each roadmap is published both as an idea of what is upcoming in `ansible-core`, and as a medium for seeking further feedback from the community.

You can submit feedback on the current roadmap in multiple ways:

- Edit the agenda of an Core Team Meeting (preferred)
- Post on the `#ansible-devel` chat channel (using Matrix at ansible.im or using IRC at irc.libera.chat)
- Email the ansible-devel list

See *Ansible communication channels* for details on how to join and use the email lists and chat channels.

1.44.1 Ansible-core 2.15

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this ansible-core release.

Note: Dates subject to change.

Development Phase

The `milestone` branch will be advanced at the start date of each development phase.

- 2022-10-17 Development Phase 1
- 2022-12-19 Development Phase 2
- 2023-02-20 Development Phase 3

Release Phase

- 2023-03-31 Feature Freeze (and `stable-2.15` branching from `devel`) No new functionality (including modules/plugins) to any code
- 2023-04-03 Beta 1
- 2023-04-24 Release Candidate 1
- 2023-05-15 Release

Note: The beta and release candidate schedules allow for up to 3 releases on a weekly schedule depending on the necessity of creating a release.

Release Manager

Ansible Core Team

Planned work

- Proxy prompting over queue from forks

Delayed work

The following work has been delayed and retargeted for a future release:

- Data Tagging

1.44.2 Ansible-core 2.14

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this ansible-core release.

Note: Dates subject to change.

Development Phase

The `milestone` branch will be advanced at the start date of each development phase.

- 2022-05-02 Development Phase 1
- 2022-06-27 Development Phase 2
- 2022-08-08 Development Phase 3

Release Phase

- 2022-09-19 Feature Freeze (and `stable-2.14` branching from `devel`) No new functionality (including modules/plugins) to any code
- 2022-09-26 Beta 1
- 2022-10-17 Release Candidate 1
- 2022-11-07 Release

Note: The beta and release candidate schedules allow for up to 3 releases on a weekly schedule depending on the necessity of creating a release.

Release Manager

Ansible Core Team

Planned work

- Implement sidecar docs to support documenting filter/test plugins, as well as non Python modules
- Proxy Display over queue from forks
- Move handler processing into new PlayIterator phase to use the configured strategy
- Convert FieldAttribute to data descriptors to avoid complex meta classes
- Drop Python 3.8 support for controller
- Enforce running controller code with the Python locale and filesystem encoding set to UTF-8

Delayed work

The following work has been delayed and retargeted for a future release:

- Data Tagging

1.44.3 Ansible-core 2.13

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this ansible-core release.

Note: There is no Alpha phase in 2.13.

Note: Dates subject to change.

Development Phase

The `milestone` branch will be advanced at the start date of each development phase.

- 2021-09-27 Development Phase 1
- 2021-12-13 Development Phase 2
- 2022-02-14 Development Phase 3

Release Phase

- 2022-03-28 Feature Freeze (and `stable-2.13` branching from `devel`) No new functionality (including modules/plugins) to any code
- 2022-04-11 Beta 1
- 2022-04-25 Beta 2 (if necessary)
- 2022-05-02 Release Candidate 1
- 2022-05-16 Release

Release Manager

Ansible Core Team

Planned work

- `ansible-doc` extended dump support for devtools integration
- `ansible-galaxy` CLI collection verification, source, and trust
- `jinja2` 3.0+ dependency
- Consolidate template handling to always use `jinja2` native
- Drop Python 2.6 support for module execution
- Update the collection loader to Python 3.x loader API, due to removal of the Python 2 API in Python 3.12
- Modernize python packaging and installation

Delayed work

The following work has been delayed and retargeted for a future release

- Data Tagging
- Implement sidecar docs to support documenting filter/test plugins, as well as non Python modules

1.44.4 Ansible-core 2.12

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this ansible-core release.

Note: There is no Alpha phase in 2.12.

Note: Dates subject to change.

- 2021-09-24 Feature Freeze (and `stable-2.12` branching from `devel`) No new functionality (including modules/plugins) to any code
- 2021-09-27 Beta 1
- 2021-10-04 Beta 2 (if necessary)
- 2021-10-18 Release Candidate 1
- 2021-10-25 Release Candidate 2 (if necessary)
- 2021-11-08 Release

Release Manager

Ansible Core Team

Planned work

- Bump the minimum Python version requirement for the controller to Python 3.8. This will be a hard requirement.
- Deprecate Python 2.6 support for managed/target hosts. The release of `ansible-core==2.13` will remove Python 2.6 support.
- Introduce split-controller testing in `ansible-test` to separate dependencies for the controller from dependencies on the target.
- Extend the functionality of `module_defaults` `action_groups` to be created and presented by collections.

Delayed work

The following work has been delayed and retargeted for a future release

- Implement object proxies, to expose restricted interfaces between parts of the code, and enable deprecations of attributes and variables.

1.44.5 Ansible-core 2.11

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this ansible-core release.

Note: There is no Alpha phase in 2.11.

Note: Dates subject to change.

- 2021-02-12 Feature Freeze No new functionality (including modules/plugins) to any code
- 2021-03-02 Beta 1
- 2021-03-15 Beta 2 (if necessary)
- 2021-03-29 Release Candidate 1 (and `stable-2.11` branching from `devel`)

- 2021-04-12 Release Candidate 2 (if necessary)
- 2021-04-26 Release

Release Manager

Ansible Core Team

Planned work

- Rename `ansible-base` to `ansible-core`.
- Improve UX of `ansible-galaxy collection` CLI, specifically as it relates to install and upgrade.
- Add new Role Argument Spec feature that will allow a role to define an argument spec to be used in validating variables used by the role.
- Bump the minimum Python version requirement for the controller to Python 3.8. There will be no breaking changes to this release, however `ansible-core` will only be packaged for Python 3.8+. `ansible-core==2.12` will include breaking changes requiring at least Python 3.8.
- Introduce split-controller testing in `ansible-test` to separate dependencies for the controller from dependencies on the target.

1.44.6 Ansible-base 2.10

Release Schedule

Expected

PRs must be raised well in advance of the dates below to have a chance of being included in this ansible-base release.

Note: There is no Alpha phase in 2.10.

Note: Dates subject to change.

- 2020-06-16 Beta 1 **Feature freeze** No new functionality (including modules/plugins) to any code
- 2020-07-21 Release Candidate 1 (bumped from 2020-07-14)
- 2020-07-24 Release Candidate 2

- 2020-07-25 Release Candidate 3
- 2020-07-30 Release Candidate 4
- 2020-08-13 Release

Release Manager

@sivel

Planned work

- Migrate non-base plugins and modules from the `ansible/ansible` repository to smaller collection repositories
- Add functionality to ease transition to collections, such as automatic redirects from the 2.9 names to the new FQCN of the plugin
- Create new `ansible-base` package representing the `ansible/ansible` repository

Additional Resources

The 2.10 release of Ansible will fundamentally change the scope of plugins included in the `ansible/ansible` repository, by moving much of the plugins into smaller collection repositories that will be shipped through https://galaxy.ansible.com/

The following links have more information about this process:

- https://groups.google.com/d/msg/ansible-devel/oKqgCeYTs-M/cHrOgMw8CAAJ
- https://github.com/ansible-collections/overview/blob/main/README.rst

PYTHON MODULE INDEX

INDEX