The Essential Guide To Load-Balancing Minecraft Servers With Kong Gateway

Dev Spotlight is my company. We create tech content for tech companies. Email at

It's time for some fun. You can tell your family and colleagues that you are doing research or experimenting with new tech, but don't let them know that you're playing Minecraft.

Let's say you're organizing a Minecraft class for local STEM students. You need to run your own Minecraft servers to ensure a kid-friendly multiplayer environment, restricted only to your students. minecraft servers You won't have enough servers so you will need to run two servers simultaneously. Your load balancer will handle sending students to Server A and Server B depending on their load.

In this article we will explore port forwarding with Kong Gateway and load balancing. We're going to do this by spinning up multiple Minecraft servers, and then placing Kong Gateway in front of these upstream services to handle port forwarding and load balancing.

Before we dive in, let us briefly review some key technology concepts.

Key Concepts

Port forwarding involves receiving network requests on one port of a machine, and forwarding those requests onto another port. A router, firewall or API gateway usually handles this task. A web server might listen on port 3000 while a database server may be listening on port 5500. Your API gateway would listen for outside requests. The gateway would forward all requests addressed to port80 to your webserver at ports 3000 and 3000. In the meantime, port 5432 requests will be forwarded by your gateway to your web server at Port 3000.

Load Balancing

Load balancing refers to the process of distributing multiple requests on a server in a balanced way across many replicas of that server. This is usually done by a piece of hardware or software known as a load balancer. The outside world doesn't know that there are multiple copies of a server. They think they are requesting information from one server. The load balancer distributes the request load to ensure that no one server is overwhelmed. The load balancer ensures that requests go only to healthy nodes in the event of a complete failure of a replica.

Kong Gateway

Kong Gateway is an API gateway layer that sits in front upstream services and can perform load balancing and port forwarding tasks. Whether those upstream services are web servers or databases or even Minecraft game servers, Kong is the front-door greeter to all requests. Kong Gateway can manage traffic control and authentication as well as request transformations, analytics, logging, and log.

TCP Stream Support

Minecraft, unlike other web servers and databases, requires a connection between the Minecraft client (the user) and the server. Rather than expecting stateless HTTP requests, we'll need to handle TCP connections with streaming data. TCP streaming is supported fully by Kong Gateway.

Our project approach

We'll walk you through this project step by step.

1. Spin up a single, local Minecraft server without any port forwarding. 2. Spin up a Minecraft server on a non-default port, configuring Kong Gateway to port forward requests to that server. 3. Spin up two Minecraft servers on different ports, configuring Kong Gateway to load balance and port forward connection requests.

As you can see we will start simple and slowly increase complexity.

What you'll need to get started

To complete this mini-project, you don't need to be familiar with Minecraft. Since it's easiest to spin up Minecraft servers within Docker containers, basic familiarity with Docker may be helpful.

Docker Engine needs to be installed on the local machine. To verify that our project is successful, you will need to install the Minecraft client and log in to the game as a paid owner. Minecraft's free trial does not allow you to connect to multiplayer servers. This is what we will be using for our project.

Are you ready to do it? Here we go!

Step 1: Single Minecraft Server with Default Port

In this first step, we want to spin up a single Minecraft server on our local machine. The default port will be used for the server. Next, we'll connect the game client to the server. It's simple to deploy the Minecraft server as a Docker container, with the Docker image found here.

In a terminal, run the following command to pull down the image server and spin it up into a container.

As our container starts up, it downloads the Docker image for the Minecraft server. Once the image has been downloaded, the server is started up and the log messages are displayed. Here are the flags and options that we gave to docker run in this command:

-p indicates a port on a host (your local machine) that Docker should associate with a container port. In this case, the container's port number 25565 will be mapped to our local machine's Port 25000. By default, Minecraft servers run on port 25565. Regardless of the port on your host, you will bind to the container's port 25655.

-e EULA=true is an environment variable that Docker container needs when starting up the server. The Minecraft server application requires that you accept the EULA upon startup. This environment variable can also be done using Docker.

- Lastly, we specify the name of the Docker image (on DockerHub), which contains the Minecraft server.

Now that our server is up, let's connect to localhost:25000. Open up the Minecraft Launcher client and click on "Play".

The actual Minecraft game should launch. Click on "Multiplayer" for more game options.

Next, click on "Direct Connection".

For server address, enter localhost:25000. Our local port 25000, of course, is bound to the container running our Minecraft server. Finally, we click "Join Server".

And... we're in!

If you look back at the terminal with the docker run command, you'll recall that it continues to output the log messages from the Minecraft server. It could look something like this.

The server reports that a new player has joined the game (my username is familycodingfun). Our single game server setup is complete. Let's now add Kong Gateway to the mix. We'll close the game now and kill the Docker container that we have with the server.

Step 2: Minecraft Server with Kong Gateway and Port Forwarding

Next, we'll put Kong Gateway in front of our Minecraft server and take advantage of port forwarding. If you were running a private network, you might forbid requests from outside the network to reach your Minecraft server port. You might also expose one port to Kong that Kong listens to. Kong, as the API gateway, would listen to requests on that port and then forward those requests to your Minecraft server. Doing so ensures that any requests that want to go to a Minecraft server must go through Kong first.

Although we will be working with localhost, this type of port forwarding will be done through Kong. Just like in our previous step, we want our Minecraft server to run on port 25000. Kong will listen to port 20000 while we wait. Kong will take TCP connection requests on port 20000 and forward them to the Minecraft server at port 25000.

Install and Setup Kong

The first step is to install Kong Gateway. The steps for each setup will vary. After installing Kong we will need to set up the initial configuration. You'll find a template file called kong.conf.default in your /etc/kong directory. This file will be copied and renamed kong.conf. It will be used by Kong to configure its startup.

We will need to make these three edits in kong.conf

The stream_listen configuration tells Kong that it will listen for streaming TCP traffic. This tells Kong to listen on port 20000. For the needs of this mini project, we can configure Kong using its DB-less and Declarative configuration style. Kong will not need to use a database (database = off), and all of our configurations for port forwarding and load balancing will be stored in a single YAML file. This is the path for declarative_config files that we've given above.

Write Declarative Configuration File

Before we start up Kong, we need to write that minecraft-kong.yml file with our port forwarding configuration. Open minecraft.yml from a project folder.

This file will declare a new Service entity called Minecraft Server-A. The server uses the TCP protocol, listening on localhost port 25000, so we set these values together as the service's url. Next, we define a route for the service. This associates our server with a URL path, or an incoming connection destination, that Kong will listen to. We give Kong a route name, telling Kong that it will listen for requests using TCP/TLS to the destination specified in our kong.conf.

Start Up Minecraft Server and Kong

We have all the configuration needed for this step. Let's start up our Minecraft server in Docker. Remember, we want our host (our local machine) to be ready on port 25000, binding that port to the standard Minecraft server port of 25565 on the container:

It might take some time to execute this command as the server boots up.
Now, we'll open Kong in a separate terminal.

~/project$ sudo kong start

Once our server is up and running, we return to our game client and choose "Multiplayer" to try to establish a "Direct Connection". We know that we can connect directly with localhost:25000 because that's actually the host port bound for the container's Port; but we want to test Kong’s port forwarding. We want to connect with the game server on localhost.20000 pretending we are a casual user who doesn't know that port 20000 is a port forwarding gateway.

Click on "Join Server." You'll be able to enter the Minecraft world if your connection succeeds like Step 1. Our TCP connection request to localhost:20000 went to Kong Gateway, which then forwarded that request to port 25000, our actual Minecraft server. We have port forwarding up and running!

Step 3: Load-Balancing Two Minecraft Servers

We will spin up two Minecraft servers for the final step in our mini-project, listening on ports

25000 and 26000. Previously, when we only had one Minecraft server, Kong would naturally forward TCP requests at port 20000 to that sole Minecraft server's port. Now, with two Minecraft server ports to choose from, we'll need to use port forwarding and load balancing. Kong Gateway will take TCP connection requests that come to port 20000 and distribute connections evenly between Minecraft Server A and Minecraft Server B.

Start Up Minecraft Servers

If you haven't done so already, terminate the single Minecraft server that was running in the previous step. We'll start all over again in a clean state by spinning up each server from its own terminal window. In the first terminal, run the Docker Container for Server A. Bind Server 25000 to the container’s port 25565.

~/project$ docker run -p 25000:25565 -e EULA=true itzg/minecraft-server

Next, we will open Server B in a separate terminal and bind the host's port 26000 the container's ports 25565.

~/project$ docker run -p 26000:25565 -e EULA=true itzg/minecraft-server

Servers A, and B, are now available at ports 25000 to 26000, respectively.

Edit Declarative Configuration File

Next, we need to edit our declarativeconfig file (minecraft_kong.yml), configuring Kong for load balancencing. Modify your file to reflect these changes:

Let's take a look at what we did. First, we created an upstream object (arbitrarily titled Minecraft Servers), which serves as a virtual hosting server for load balancing between multiple services. This is exactly what we need. We added two Target Objects to our upstream service. Each target has an address with host and port; in our case, our two targets point to localhost:25000 (Minecraft Server A) and localhost:26000 (Minecraft Server B). Next, we assign a weight to each target. This is what the load balancer uses for load distribution. Even though we've explicitly set the weights evenly to 100, the default for this optional configuration is 100.

Next, we declared our Service Object, which in this case is our load balancer service. Requests that meet the routes we have established will be sent to Minecraft-Servers host, which is our load balancing upstream object. Similar to the last step, we created a route to

tell Kong Gateway to listen out for TCP/TLS queries destined for 127.0.0.1.20000.

Restart Kong

Our Kong configuration has changed. We must restart Kong to allow the changes to take affect.

~/project$ sudo kong restart

At this point, everything is up and running. We have our two Minecraft servers (Server A and Server B) running in Docker containers in two separate terminal windows. Kong is configured on port 20000 to listen for TCP, forwarding these requests to our loadbalancer. This distributes connections across the two servers.

Open the Minecraft game client once again. Similar to the previous steps we will connect to the multiplayer server localhost:20000 directly. As you connect, keep an eye on your two server terminal windows. You will see messages for Server A and Server B as you connect and disconnect from the game.

And just like that, we have set up our load balancer to distribute connection requests across our two Minecraft servers!

Ready to (Play!) Work

We have now reached a certain level of complexity in our mini-project.

1. We started by simply spinning up a single Minecraft server in a Docker container, using port 25000 for accepting game client connections. 2. Next, we installed Kong Gateway on our single server to perform port forwarding. Kong listened on port 20000 for game client connections, forwarding those requests to the port on our host where the Minecraft server was accessible. 3. Lastly, we set up two Minecraft servers to run concurrently. Then, we configured Kong Gateway to act as a load balancer. Kong Gateway then listened on port 20000 to game client connection requests, redirecting them through its load-balancing service.

From here, you have many opportunities for adding complexity. You can add additional game servers. For example, if some servers run on machines that can handle a heavier load of connections than others, then you can configure the load balancer to distribute the load unevenly. To ensure that only healthy servers are forwarding requests to Kong's load-

balancer, you can create health check rules. There are many load balancing options available, including the "round-robin" default strategy.

We've had some fun and learned important concepts and tools along the way. Kong Gateway makes load balancing, port forwarding simple and easy. These features are powerful even though they are so simple. Now that you've got a handle on it, it's time to get to work and face the Ender Dragon.