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 [email protected]

It's time for some fun. Tell your family, friends, and coworkers that you're conducting research, or trying out new tech. But don't let your family see that you're playing Minecraft!

Here's what it looks like: You are organizing a Minecraft class to teach local STEM students. You need to run your own Minecraft servers to ensure a kid-friendly multiplayer environment, restricted only to your students. 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.

This article will discuss port forwarding and load balance with Kong Gateway. 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 refers to receiving network requests on a specific port of a computer and forwarding them to another port. This task is usually handled by a router, firewall, or API gateway. For example, you might have a web server listening on port 3000 and a database server listening on port 5000. Your API gateway would listen for requests from outside your network. 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 is the task that distributes multiple requests to a server evenly across multiple replicas. A specific piece of hardware or software called a load balancer usually handles this. The outside world does not know that there are multiple servers running. They believe they're making requests to a single server. The load balancer, however, distributes the request load to prevent any one server from being overwhelmed. The load balancer ensures requests only reach healthy nodes in case of a total failure of the 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. Kong Gateway is the front door greeter to all requests, no matter if those upstream services are web server or database servers or Minecraft game servers. Kong Gateway is capable of managing traffic control as well as authentication, request transformations (analytics), and logging.

TCP Stream Support

Our Minecraft project is different from other web servers or databases in that Minecraft requires an established connection between the Minecraft client (the player) and the server. Instead of accepting stateless HTTP request, we'll be handling TCP connections that use streaming data. TCP streaming is fully supported by Kong Gateway.

Our project approach

We'll walk you through the 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 clearly see, we'll start with simplicity and gradually build up complexity.

What you'll need to get started

This mini-project does not require you to have a lot knowledge of Minecraft. Since it's easiest to spin up Minecraft servers within Docker containers, basic familiarity with Docker may be helpful.

Docker Engine will need to be installed on your 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 this? Let's 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. We'll use the default port for the server, and then we'll connect our game client to the server. It's simple to deploy the Minecraft server as a Docker container, with the Docker image found here.

We'll use a terminal window for this command to pull the server image down and spin it up inside 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's an explanation of flags and options we provided to docker run in our command:

-p is a port on your local machine (host port) that Docker should connect to a container port. In this case, our local machine's port 25000 will point to the container's port 25565. By default, Minecraft servers run on port 25565. Typically, you will always bind to the container's port 25565, regardless of the port on the host that you choose to use.

-e EULA=true specifies the environment variable that Docker container must use to start up the server within the container. 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.

Let's start our server and see if it can connect to the localhost.25000 server. Open up the Minecraft Launcher client and click on "Play".

The actual Minecraft game should start. For game options, click on "Multiplayer".

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 on the "Join Server" button.

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 might look something like this:

The server informs me that a new user (my username familycodingfun) has joined our game. Our single game server setup is complete. Let's add Kong Gateway as well as port forwarding. 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. At the same time, you might expose a single port on which Kong listens. 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.

We'll be working in localhost but we'll setup port forwarding through Kong. Just like in our previous step, we want our Minecraft server to run on port 25000. Meanwhile, Kong will listen on port 20000. Kong will take TCP connection requests on port 20000 and forward them to the Minecraft server at port 25000.

Install and Setup Kong

Install Kong Gateway is the first step. The steps of installation will vary depending on the configuration. After installing Kong, we'll need to set up the initial configuration file. In your /etc/kong/ folder, you will find a template file named kong.conf.default. This file will be copied to kong.conf and renamed as kong.conf. Kong will use this file for its startup configuration.

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

The stream_listen configuration tells Kong how to listen for streaming TCP data. We tell Kong to listen at 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 a database (database=off), as all configurations for port forwarding, load balancing, and load balancing can be stored in one YAML file. This is the path to declarative_config that we have set above.

Write Declarative Configuration File

Before we can launch Kong, we will need to create the minecraft.yml.yml file that contains our port forwarding configuration. Open minecraft.yml, a file that corresponds to the path you specify above, in a project folder.

This file will declare a new Service entity called Minecraft Server-A. These values will be used as the service's web address. The server uses TCP protocol. It is listening on localhost port 250000. Next, we define a Route for the service, which associates our service with a URL path or an incoming connection destination that Kong will listen for. 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:

This command may take a while to execute as the server is starting up.
Now, we'll open Kong in a separate terminal.

```
~/project$ sudo kong start
```

With our server up and running, we go back to our game client and, just like above, choose "Multiplayer" and try to establish a "Direct Connection" with a game server. 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 to the supposed game server on localhost:20000, pretending that we're the casual user who is unaware that port 20000 points to 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
```

Now, Servers A & B are available at ports 25000 and 26000, respectively.

Edit Declarative configuration File

Next, we will edit our declarative file (minecraft-kong.yml), configuring Kong to load balance. Edit your file to reflect the following:

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. That's exactly what we need. To our upstream service, we added two Target Objects. 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). The load balancer then uses this weight to distribute load. Even though we've explicitly set the weights evenly to 100, the default for this optional configuration is 100.

Next, we created our Service Object. It is our load balancer. Requests that comply with the routes will be forwarded the Minecraft-Servers host. FLASHANTS This upstream load balancer object will forward requests that satisfy these routes. 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 to listen for TCP port 20000. This forwards those requests to our loadbalancer. We distribute connections across our two servers.

Open the Minecraft game client once again. Similar to the previous steps, we will try to connect directly to the multiplayer server located at localhost:20000. Keep an eye on the two terminal windows of your server as you connect. As you connect to the server, disconnect and reconnect repeatedly, you will sometimes see a connection message for Server A and a message for Client B.

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

Ready to (Play) Work

Let's recap: We gradually increased in complexity for 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. Next, we set Kong Gateway up as a load balancer. Kong Gateway then listened on port 20000 to game client connection requests, redirecting them through its load-balancing service.

This is where you can add complexity. You can add more 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. You can also choose from a number of load balancing algorithms, other than the default "round-robin".

So far, we've had some fun, and have learned some important concepts and tools. It's easy to set Kong Gateway up for load balancing or port forwarding. Yet, even with such ease, these features are extremely powerful. Now that you have a good grasp of the basics, it's time for you to tackle the Ender Dragon.