**Name of Title:** Learning Nginx

**Video Name:** 04\_03 Configuring a Load Balancer

**Estimated Length:**

**Author Name:** Michael Jenkins

**Chapter\_Section\_Video:**

**Video Objective:**

At the end of this video the learner will understand how an upstream HTTP source is used to configure load balancing.

**Script:**

SLIDE: 04\_04 Configuring a Load Balancer

Proxying and load balancing are similar functions for nginx. But you have no idea how similar they are! In fact, they’re pretty much the same. The only difference is the number of servers in the upstream and the method used to connect to each of the servers in turn.

The documentation at nginx.org gives more details on using nginx as a load balancer.

Let’s take a look at the different load balancing methods nginx uses.

SLIDE: Load Balancing Directives

Nginx uses three methods to connect to upstream servers: Round Robin, Least Connections, and IP Hashing. These methods are indicated with directives applied within an upstream block.

The default method is Round robin. And there’s not directive to use this method. Requests are sent to each upstream server one at a time with no preference given to the server. This is a easy, reasonable method for evenly distributing a load over several servers.

With the leasts connections method, Nginx considers the number of connections an upstream server is already processing. The server with the fewest connections will be preferred for every new request that comes in.

The IP Hash method is used for session persistence. This method considers the IP address of the client making the request and then chooses an upstream server to handle the request. If the client’s IP address stays the same, the same upstream server will handle any future requests made by the client.

The weight directive is used to influence the other methods. Since each server is considered evenly in the round robin method and by connections in the least connections method, a weight can be applied to a server to give it a higher preference. Take for example a group of upstream servers where one server has twice the CPU or memory compared to the other servers. We could give that server a higher weight so nginx can route more traffic to it.

Now let’s take a look at how we implement each of these load balancing methods.

**BREAK?**

TERMINAL

Vagrant up

Sudo su -

I’m connected to our development VM as the root user i’ll edit the file etc nginx upstream.conf.

vim /etc/nginx/conf.d/upstream.conf

We already have one upstream defined and it includes just one server. If we make an upstream with more than one server, nginx will treat any proxy pass connections to it as a load balancer. Pretty neat, right?

So let’s make another upstream block call it round robin:

upstream roundrobin {

}

I’m calling this block round robin to describe the way nginx will connect to the servers in this block. Again, calling the block round robin isn’t a requirement, it's just a naming convention I’m using for demonstration purposes.

Now let’s add our servers. I’ll add three servers and you’ll recognize one of them from the other upstream block in this configuration:

upstream roundrobin {

server 127.0.0.1:7001;

server 127.0.0.1:7002;

server 127.0.0.1:7003;

}

So I’ve added three servers, all running on the localhost and all listening on different ports. One of these servers, the one listening on 7001, is also in the app\_server\_7001 upstream block. Nginx allows servers to be used in multiple upstreams. This is useful in cases where we have upstream servers that provide different functionality.

Now let’s add the location that connects to this upstream:

location /roundrobin {

proxy\_pass http://roundrobin/;

}

Again, this is exactly the same as our proxy location, the only difference is in the upstream we’re connecting to.

Now let’s save the file and test this configuration:

SAVE FILE

Nginx -t

Systemctl reload nginx

In the browser, we’ll connect to the /roundrobin location on the VM’s IP. Each server in our load balancer should respond with the port its listening on one after the other.

BACK IN THE BROWSER

<http://192.168.0.3/roundrobin>

The first response we get is 7001 which is good. Now if we refresh the page a few times, we should see the port for each of the three app servers:

REFRESH PAGE A FEW TIMES

And indeed, each app server is responding in turn.

Now let’s configure examples for the other load balancing methods.

TERMINAL

vim /etc/nginx/conf.d/upstream.conf

In the upstream configuration file, I’m just going to copy the roundrobin upstream block and paste it three times.

This first copy I’ll rename as least conn; the second one as iphash, and the third one as weighted. Again these are requirements for the load balancing method to work, I’m just naming these upstream blocks this way for discussion purposes.

To tell nginx we need to indicate the method we want to use with each upstream. So I’ll add least\_conn to the least connections upstream, ip\_hash to the IP Hash upstream, and I’ll put a weight of 2 on the first server in the weighted upstream.

UPDATE FILE

Now let’s create a location for each of these new upstreams. And again, I’ll just copy the round robin location and change it for the new methods we’re demonstrating.

UPDATE FILE

Now let’s save the file and test this configuration:

SAVE FILE

Nginx -t

Systemctl reload nginx

And we can go to the browser not to test the load balancing methods.

http://192.168.0.3/leastconn

The least connections method is kind of hard to test in our demo environment because we can’t create enough connections to see which upstream server is getting the most connections. So if I refresh the page a few times, it looks more like a round robin method is being used. This is reasonable since round robin is the default load balancing method.

Let’s try the IP Hash method.

<http://192.168.0.3/leastconn>

Now this test makes more sense. We’re using the same IP address to connect to our demo server so nginx will keep routing us to the same server. So if I keep refreshing the page, I’ll keep seeing 7001.

REFRESH PAGE A FEW TIMES

Now let’s look at the weighted upstreams.

<http://192.168.0.3/weighted>

In this case, we have three servers in the upstream with the 7001 server having a weight of 2. We’re also using the default round robin method. So what we should see is nginx connect to 7001 twice as much as the 7002 and 7003 servers. Let’s give it a shot.

On the first load, we get 7001. If I load again, we still see 7001 but the time changed. Now if I refresh again, we see 7002. And one more time, now we see 7003. And again: 7001, 7001, 7002, and 7003. So sure enough, nginx is giving more requests to the 7001 server since it has the higher weight.

Now we’ve seen a few different ways that nginx can be configured for load balancing.

**Exercise Files:**

**Basement:**

<https://www.youtube.com/watch?v=SpL_hJNUNEI>

<http://nginx.org/en/docs/http/load_balancing.html>

I’ll be using the exercise files for this lesson and, if you have access to them, I suggest you follow along with them as well. Using the Vagrantfile for this lesson will build a development VM with nginx installed and three app server that listen on ports 7001, 7002, and 7003.

If you don’t have access to the exercise files, don’t worry. You can follow along with a VM running Ubuntu 18.04 LTS. You just need to install nginx.

In this lesson we’ll continue discussing upstreams and how they’re used to set up load balancers in nginx.