**Name of Title:** Learning Nginx

**Video Name:** 04\_02 Configuring a Reverse Proxy

**Estimated Length:**

**Author Name:** Michael Jenkins

**Chapter\_Section\_Video:**

**Video Objective:**

At the end of this video the learner will be able to configure an upstream HTTP source and a reverse proxy.

**Script:**

SLIDE: 04\_02 Configuring upstreams

Now that we’ve discussed reverse proxies and load balancers, let’s take a look at how they’re implemented. The upstream module is one of the key components used to configure nginx for proxying and load balancing.

You can find more information on the upstream directive in the documentation on nginx.org.

EDITOR: Fly in <http://nginx.org/en/docs/http/ngx_http_upstream_module.html>

SLIDE: Contexts in nginx.conf 1

Before we can set up a proxy or a load balancer in nginx, we need to define the back end servers that nginx will be connecting to. This is where the upstream directive comes into play.

If we consider nginx’s configuration, there are certain contexts where we can apply directives: we can apply them at a global context that affects all of nginx; the HTTP context for items that work with the HTTP protocol; and inside the HTTP context, server and location contexts, that specifically configure virtual hosts and the URIs used to access them.

SLIDE: Contexts in nginx.conf 2

Upstreams are defined in the HTTP context. This is useful so that one upstream can be defined and then reused by multiple servers also defined in the HTTP context.

And as a side note, we refer to these resources as “upstream” because in the flow of a web request, the resources would be upstream from the nginx server and the client.

OK. Let’s go to our development server and configure our first upstream resource.

TERMINAL

Vagrant up

Sudo su -

As the root user, let’s create a configuration file call upstream.conf under /etc/nginx/conf.d:

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

Since nginx is configured to read all files in the conf.d directory that end in .conf, this is a good location to store our upstream configuration files.

Now we need to add our upstream block along with a name for it:

upstream app\_server\_7001 {

}

We’ll call this upstream block app server 7001; I’m including the 7001 just to note the port that the app server is listening on. This isn’t a requirement, but just a naming convention. You can use any name you’d like for an upstream block.

Now Inside this block, we can define the servers that are a part of this upstream. In this case, let’s set up the one server listening on port 7001.

upstream app\_server\_7001 {

server 127.0.0.1:7001;

}

The server statement in an upstream block is different than the server statement used to define a virtual host. In the case of an upstream server, we have to include the name or IP address of the server that this block connects to. We could also use a unix socket if the back end server is listening on a socket but in this case, the server is another HTTP process running on the same VM as our nginx process, so we’ll use the localhost IP address: 127.0.0.1 and also indicate the port 7001.

Hi editors this is a pickup to add to the end of chapter 4 video 2:

Now that we have our upstream defined, we need to connect to it.

***STOP***

***04\_03 STARTS HERE***

To connect to an upstream, we need to have a server block. In this case, we’ll use the same file that our upstream is defined in. Its not a requirement to have these in the same file, but we’ll do that now for simplicity.

server {

and we’ll have it listen on port 80.

listen 80;

We’ll also add a location block and name it /proxy.

location /proxy {

}

}

Now we can add the proxy with the proxy pass directive. This comes from the nginx HTTP proxy module. For more information on the proxy module, you can refer to the documentation on nginx.org.

EDITOR: FLY IN <http://nginx.org/en/docs/http/ngx_http_proxy_module.html>

server {

listen 80;

location /proxy {

**proxy\_pass http://app\_server\_7001/;**

}

}

OK let’s discuss this for a moment. This server we just defined will listen on port 80. If a requests comes in on /proxy, the server will use the location that matches so it will send the traffic into the /proxy location for processing. With the proxy\_pass directive, the request will be sent the the HTTP resource defined in the upstream named app\_server\_7001. Which in this case is our process listening on 7001. Any response generated by the upstream is then returned to nginx to be sent back to the client.

I’m also going to add a comment here just as a reminder that the trailing slash on the proxy pass statement is key.

server {

listen 80;

location /proxy {

**# Trailing slash is key!**

proxy\_pass http://app\_server\_7001/;

}

}

Without the trailing slash, nginx would try to connect to a location named “slash proxy” on the backend server. In most cases, you’ll just want to proxy to the root of the back end server and let the application route the request as needed.

OK enough talking about this configuration! Let’s see how it works. We’ll save the file, test the configuration, and then reload it. The app server we’ll be connecting to just returns the port that its listening on and the time the page was served so we should see 7001 when we load /proxy on the VM’s IP address.

SAVE FILE

Nginx -t

Systemctl reload nginx

Now we can open the browser to see the result.

BACK IN THE BROWSER

<http://192.168.0.3/proxy>

Cool! It works. By hitting the slash proxy location on our demo VM, we’re successfully proxying through nginx to the app server on port 7001. If we refresh the page a few times we can see that the time changes as well indicating we’re getting new content from the app server.

**Conclusion:**

Type your conclusion statement here.

**Script and Media:**

Break the script up into parts and align it with any media (slides, web, CLI, etc.)

| **Part** | **Script** | **Media** |
| --- | --- | --- |
|  |  |  |

**Exercise Files:**

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

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

**Basement:**

upstream **backend** {  
 hash $remote\_addr consistent;  
  
 server backend1.example.com:12345 weight=5;  
 server backend2.example.com:12345;  
 server unix:/tmp/backend3;  
  
 server backup1.example.com:12345 backup;  
 server backup2.example.com:12345 backup;  
}

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 start a simple app server that listens on port 7001.

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 the next lesson, we’ll expand on our upstream and proxy pass configurations to configure several different types of load balancers.