**Name of Title:** Learning Nginx

**Video Name:**

**Estimated Length:**

**Author Name:** Michael Jenkins

**Chapter\_Section\_Video:**

**Video Objective:**

At the end of this video the learner will be able to use the location directive to match URLs to paths on the server’s file system.

**Introductory Statement:**

Type your introductory statement here.

**Speaking Points:**

1. Location directive
2. Location “/” directive
3. Create custom 400 and 500 pages
4. Configure locations for 400 and 500 pages
5. Test and demo configuration

**Script:**

TITLE SLIDE:

Let’s discuss how we can make the configuration for our demo site more robust by adding location directives.

SLIDE:

URI - http://example.com/request/.../...

server {

location *[modifier]* location\_definition {

…

location {

}

}

}

Location directives allow us to extend our configuration based on the URI, or uniform resource indicator, that makes up the request being processed by the server.

When we define a location, we use the location keyword, an optional modifier, and a required location definition. Location directives are formatted as blocks and are defined inside server blocks. They can also be nested inside other location blocks.

Inside the location block, we can include other directives. Specifically, we can include just about any directive that can be defined inside a server block. This makes locations very useful since we can configure nginx to process different requests in ways similar to server blocks without having to create additional servers.

When Nginx processes locations, it uses a series of matches to determine the best one to use, first considering exact matches, then locations with prefixes, and then locations with regular expressions.

For more information on locations, take a look at the documentation on nginx.org.

Switch to terminal! :D

Let’s continue by going back to our development server as the root user and editing the configuration for our demo site:

sudo su -

vim /etc/nginx/conf.d/wisdompetmed.local.conf

Now let’s add a new location block for an empty request or just the “slash” character. This will correspond to the root directory of our site.

location / {

}

Inside this block we’ll add another directive: try files.

location / {

try\_files

}

The try\_files directive gives nginx a list of files or directories to look for relative to the location. The first file or directory that matches gets processed. If no items in the list match, then the last item in the list is used as a URI or an error code.

For this location, we’ll use the $uri variable to first look for a file that matches the URI being processed.

location / {

try\_files $uri

}

Then we’ll add the URI variable followed by a slash to test for any directories that might match.

location / {

try\_files $uri $uri/

}

And lastly, we’ll add “=404” to tell nginx to serve a 404 error if no file or directory matches the URI being requested.

location / {

try\_files $uri $uri/ =404;

}

Let's add another location for the images directory:

location /images {

autoindex on;

}

The autoindex directive will let the browser list the files in that directory. This means we’ll be able to see the files in that directory from a browser. This behaviour is turned off by default.

And now let's add a couple more directives and locations for custom error pages:

error\_page 404 /404.html;

location = /404.html {

internal;

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

internal;

}

When nginx can’t find any content to respond to a request, it displays a 404 error page. And if there are any server side issues, nginx will display a 500 error page. In these cases, we’ve added an error page directive to tell nginx to use our custom pages instead of the defaults.

For these custom pages, we’ve also added a location directive with the equal sign, also known as the exact match modifier. And inside the location block, we’ve added the internal directive which tells nginx to process any redirects to the custom 404 page as an internal redirect. Both of these -- the exact match modifier and the internal redirect -- will help nginx serve our custom error page more quickly.

Testing the 404 page will be easy, we can just request a page that we know our site isn’t set up to server. But since we don’t have any server side processing in place, we need to add a specific location to test our 500 page. To do this, we can create a fake fastcgi\_pass that we know will fail.

location = /500 {

fastcgi\_pass unix:/this/will/fail;

}

OK, Save the file and exit. And now we can test the configuration with nginx dash t.

nginx -t

The configuration looks good so we can load it with systemctl:

systemctl reload nginx

Now let’s test the locations we just added.

OPEN BROWSER

First, let's check the site. If we refresh the page we see our site is loading properly and still looks good.

Now let’s check the images location by browsing to slash images.

<http://192.168.0.3/images>

Because we used the autoindex directive, we can see the content of this directory.

Now let’s check our custom 404 page. This can be checked by just going to a location that we know isn’t defined in our site, like missing:

[http://192.168.0.3/missing](about:blank)

And there’s our custom 404 page with a cute little chicken. Now let’s test the 500 location:

<http://192.168.0.3/500>

As expected, our custom 500 page is displayed.

**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:**

**Basement:**

With locations, we can handle rcontent There are many uses for location directives including We can uses location directives to determine what content gets served in response to a request.

I’m starting at with a VM that has nginx configured and our demo site up and running.

Now l

EDITOR: Fly in the URL <http://nginx.org/en/docs/http/ngx_http_core_module.html#location>

If you have access to the exercise files, you’ll find the Vagrantfile and server configuration file I’m using.

Now Let’s take a look at the location modifiers and how they’re applied to location definitions.

SLIDE:

| **Modifier** | **Application to Location Definitions** |
| --- | --- |
| None | The location definition is interpreted as a prefix for the URI |
| = | The URI must be an exact match to the location definition |
| ~ | The location definition used as a case sensitive regular expression |
| ~\* | The location definition used as a case insensitive regular expression |
| ^~ | If the longest prefix matches, then no regular expressions are checked |

If a modifier is present, it changes the way nginx uses the location definition. Depending on the modifier, the location definition may be used as a URI prefix or a regular expression.

Because regular expressions are beyond the scope of this lesson, we’ll be looking at some examples for exact matches and prefixes.

I’m back in the terminal and I’m logged into the virtual machine running our demo site.