NGINX Core - Module M07 Labs

**** The Labs Begin on Page 2 ****

PLEASE READ THESE SYSTEM USE INSTRUCTIONS:

The lab systems default to text mode, if you want to take advantage of copy-and-paste functionality, or need to use multiple **Terminal** applications, you'll need to start the GUI Desktop environment using the following instructions.

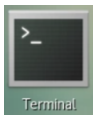
Steps to Prepare for a GUI Lab Environment:

1. Sign on to the system with the user credential **root** and password **training**.

2. Start the GUI desktop with the command:

   ```
   $ startx
   ```

3. When the Graphical Desktop appears, click on the **Terminal** app icon on the left-side of the desktop to open the Terminal.



   Note: You can resize the **Terminal** app to suit your preferences or to show more text output on the screen.

To Copy and Paste to a Lab System in GUI Mode:

1. Select the desired text from this lab guide and use **Cmd+c** (macOS) or **Ctrl-c** (Windows and Linux) to copy the text to the system clipboard.

2. Then click into the **Terminal** app in the lab system, and press Ctrl-Shift-V or select the Terminal application's **Edit** -> **Paste** menu item.

3. You may need to press it twice to insert the copied text.

   Note: When pasting into **Vim**, you MUST be in **Insert** mode first, or the keystrokes will get mis-interpreted. If you get a strange `^M` or other character first, just delete it and re-paste again.

NOTE: The **commands** you should be typing, (or copying & pasting for convenience) are displayed like the below text; (don't type or copy the **$** prompt, just the `green` text):

```
$ command
```

The **output** from commands, or the file contents shown in commands like **vim** are displayed like this:

```
Line of output
Another line of output
```

Labs for Module M07 - Proxying HTTP Requests

Labs Included

- Lab 7.1 - Setup Reverse Proxy Environment
- Lab 7.2 - Appending Paths
- Lab 7.3 - Setup the try_files Directive
- Lab 7.4 - To / or not to /

PLEASE READ!

You have been allocated **1 hour** for the labs in this section. Please leave the Virtual Lab Machine browser window/tab open after completing the first lab, then continue the section content. You will then be asked to complete the rest of the labs throughout the rest of the section.

If you let too much time lapse and the Virtual Lab Machine times out, simply restart it by clicking on the Launch button for the Lab and start again with the first lab.

Note: We have included configuration file samples in their entirety to make catching up with a particular lab easier. For example, if your lab times out and you were in a later lab, simply copy the configuration file contents into the appropriate file and continue with the next lab.

Lab 7.1: Setup Reverse Proxy Environment

Learning Objectives

By the end of the lab you will be able to:

- Configure a proxy test environment
- Configure two application configurations
- Generate traffic
- View logfiles for results

Overview

In this exercise you will configure a proxy test environment with two applications configured and then test the function, viewing the output and the log file results with the **`tail`** command.

Steps

1. Rename the **`default.conf`** configuration file:

   ```
   $ mv /etc/nginx/conf.d/default.{conf,bak}
   ```

2. Create a new configuration file named **`proxy_test.conf`**:

```
$ vim /etc/nginx/conf.d/proxy_test.conf
```

Type in or copy and paste the below configuration block to the **proxy_test.conf** file.

Note: When in Vim, press **i** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

```
# proxy_test.conf

server {
  listen 80 default_server;
  access_log /var/log/nginx/proxy.access.log combined;
  error_log /var/log/nginx/proxy.error.log info;
  root /usr/share/nginx/html;

  location /1 {
    proxy_pass http://localhost:8080;
  }
  location /2 {
    proxy_pass http://localhost:8081;
  }
}
```

Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

3. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

```
:wq
```

4. Create a new configuration file named **app1.conf**:

```
$ vim /etc/nginx/conf.d/app1.conf
```

Type in or copy and paste the below configuration block to the **app1.conf** file.

Note: When in Vim, press **i** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

```
# app1.conf

server {
  listen 8080;
  access_log /var/log/nginx/app1.access.log combined;
  error_log /var/log/nginx/app1.error.log info;
    location / {
      return 200 "this is app1\n";
    }
```

```
}
```

Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

5. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

```
:wq
```

6. Create a new configuration file named **app2.conf** file:

```
$ vim /etc/nginx/conf.d/app2.conf
```

Type in or copy and paste the below configuration block to the **app2.conf** file.

Note: When in Vim, press **i** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

```
# app2.conf

server {
  listen 8081;
  access_log /var/log/nginx/app2.access.log combined;
  error_log /var/log/nginx/app2.error.log info;
    location / {
      return 200 "this is app2\n";
    }
}
```

Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

7. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

```
:wq
```

8. Reload your NGINX configuration with

```
$ nginx -s reload
```

9. Test your NGINX configuration with:

```
$ curl localhost | grep -i "welcome"
```

Inspect the output for the phrase **Welcome to nginx!**, which should at the bottom of the command's output.

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   612  100   612    0     0   111k      0 --:--:-- --:--:-- --:--:--
119k
<title>Welcome to nginx!</title>
<h1>Welcome to nginx!</h1>
```

10. Test the **/1** location results with:

```
$ curl localhost/1
```

```
this is app1
```

As you can see, the location will show as configured.

11. Test the **/2** location results with:

```
$ curl localhost/2
```

```
this is app2
```

As you can see, the location will show as configured.

12. Now view the contents of the access logs for the three locations:

```
$ tail /var/log/nginx/{proxy,app1,app2}.access.log
```

```
==> /var/log/nginx/proxy.access.log <==
127.0.0.1 - - [01/01/2525:14:42:45 -0700] "GET / HTTP/1.1" 200 612 "-" "curl/7.29.0"
127.0.0.1 - - [01/01/2525:15:21:48 -0700] "GET /1 HTTP/1.1" 200 13 "-" "curl/7.29.0"
127.0.0.1 - - [01/01/2525:15:21:55 -0700] "GET /2 HTTP/1.1" 200 13 "-" "curl/7.29.0"

==> /var/log/nginx/app1.access.log <==
127.0.0.1 - - [01/01/2525:15:21:48 -0700] "GET /1 HTTP/1.0" 200 13 "-" "curl/7.29.0"

==> /var/log/nginx/app2.access.log <==
127.0.0.1 - - [01/01/2525:15:21:55 -0700] "GET /2 HTTP/1.0" 200 13 "-" "curl/7.29.0"
```

There should be entries from all three access logs showing successful requests. Each file is notated by the characters **==>** and the full filename.

End of Lab

Lab 7.2: Appending Paths

Learning Objectives

By the end of the lab you will be able to:

- Add a location to the configuration to show appending paths
- Test for results

Overview

In this exercise, you will change your configuration to support appending paths and then test the results, however subtle they may be.

Steps

1. Edit the **`app1.conf`** file and add a new location context named **`/test/`**:

   ```
   $ vim /etc/nginx/conf.d/app1.conf
   ```

   Note: When in Vim, press **`i`** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

   New configuration is shown in **bold** text.

   ```
   # app1.conf

   server {
     listen 8080;
     access_log /var/log/nginx/app1.access.log combined;
     error_log /var/log/nginx/app1.error.log info;
       location / {
         return 200 "this is app1\n";
       }
       location /test/ {
         return 200 "this is /test/\n";
       }
   }
   ```

   Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

2. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

   ```
   :wq
   ```

3. Reload your NGINX configuration with

```
$ nginx -s reload
```

13. Test the your configuration results with:

```
$ curl localhost/1/test
```

```
this is app1
```

The result shows the location matched is **app1**.

14. Test your appended path location with:

```
$ curl localhost/1/test/
```

```
this is app1
```

Puzzlingly, this location also shows the result for the location **app1**. This is because of the **proxy_pass** configuration prepending **/1/** to the proxied request. We'll fix this next.

15. Edit the **proxy_test.conf** and add a trailing slash **/** to the end of the **proxy_path** directive in location **/1**:

```
$ vim /etc/nginx/conf.d/proxy_test.conf
```

Note: When in Vim, press **i** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

New configuration is shown in **bold** text.

```
# app1.conf

server {
  listen 8080;
  access_log /var/log/nginx/app1.access.log combined;
  error_log /var/log/nginx/app1.error.log info;
    location / {
      return 200 "this is app1\n";
    }
    location /test/ {
      return 200 "this is /test/\n";
    }
}
```

Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

4. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

```
:wq
```

5. Reload your NGINX configuration with

```
$ nginx -s reload
```

16. Test the your configuration results with:

```
$ curl localhost/1/test
```

```
this is app1
```

The result shows the location matched is **app1**.

17.
18. Edit the **proxy_test.conf** and add a trailing slash **/** to the enod of the **proxy_path** directive in location **/1**:

```
$ xxx
```

```
output
output
```

You can see the error messages for the invalid requests reflected in the log file.

End of Lab

Lab 6.3: Customize Your Access Log

Learning Objectives

By the end of the lab you will be able to:

- Set up a custom logging format
- Change your Access logging configuration

- Test for results

Overview

In this exercise, you will customize your access log to show custom connection details and test the results.

Steps

6. Edit the **`log_test.conf`** file:

   ```
   $ vim /etc/nginx/conf.d/log_test.conf
   ```

7. Edit the **`log_test.conf`** file and add a log_format statement in the http context, and then change the access_log format from combined to custom:

   Note: When in Vim, press **`i`** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

   New configuration is shown in **bold** text, and please note that the **log_format** configuration line has no carriage returns, it's one long line.

   ```
   log_format custom "Request: $request\n Status: $status\n
   Request_URI: $request_uri\n Host: $host\n Client_IP: $remote_addr\n
   Proxy_IP(s): $proxy_add_x_forwarded_for\n Upstream_IP:
   $upstream_addr";

   server {
     listen 80 default_server;
     server_name localhost;
     access_log /var/log/nginx/test.access.log custom;
     error_log /var/log/nginx/test.error.log error;
     root /usr/share/nginx/html;
   }
   ```

   Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

8. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

   ```
   :wq
   ```

9. Reload your NGINX configuration with

   ```
   $ nginx -s reload
   ```

10. Generate some logging entries with:

```
$ curl http://localhost/test?1234
```

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.17.6</center>
</body>
</html>
```

As expected, the response to this query shows an error occurred.

19. View the access log to see the details of the unsuccessful connection:

```
$ tail /var/log/nginx/test.access.log
```

```
127.0.0.1 - - [31/Mon/2022:14:44:52 -0700] "GET /test?1234 HTTP/1.1" 404 153 "-"
"curl/7.29.0"
127.0.0.1 - - [31/Mon/2022:14:46:22 -0700] "GET / HTTP/1.1" 200 612 "-" "curl/7.29.0"
127.0.0.1 - - [31/Mon/2022:14:46:38 -0700] "GET /test?1234 HTTP/1.1" 404 153 "-"
"curl/7.29.0"
Request: GET /test?1234 HTTP/1.1
 Status: 404
 Request_URI: /test?1234
 Host: localhost
 Client_IP: 127.0.0.1
 Proxy_IP(s): 127.0.0.1
 Upstream_IP: -
```

You can see the connection message for the most recent request appears in the custom format configured in the above steps. You will find a matching request also appears in the error log.

End of Lab

Lab 6.4: JSON Format Access Log

Learning Objectives

By the end of the lab you will be able to:

- Configure JSON logging for the server block
- Test for results

Overview

In this exercise, you will configure a custom JSON logging directive, install the epel tools and the jq tool to show the log results properly.

Steps

1. Edit the **`log_test.conf`** file:

   ```
   $ vim /etc/nginx/conf.d/log_test.conf
   ```

2. Edit the **`log_test.conf`** file and add a **SECOND** log_format statement in the http context, **ABOVE** the current log_format statement, and then add a **SECOND** access_log directive **ABOVE** the existing access_log directive:

   Note: When in Vim, press **`i`** to enter insert mode, and unless you are typing the server block manually, refer to the copy-and-paste instructions on the first page of the lab.

   New configuration is shown in **bold** text. (Please format the JSON configuration as shown, for neatness and readability, we are not animals here....)

   ```
   log_format custom_json escape=json
   '{'
     '"request":"$request",'
     '"status":"$status",'
     '"client":"$remote_addr",'
     '"proxy":"$proxy_add_x_forwarded_for",'
     '"upstream": "$upstream_addr"'
   '}';

   log_format custom "Request: $request\n Status: $status\n
   Request_URI: $request_uri\n Host: $host\n Client_IP: $remote_addr\n
   Proxy_IP(s): $proxy_add_x_forwarded_for\n Upstream_IP:
   $upstream_addr";

   server {
     listen 80 default_server;
     server_name localhost;
     access_log /var/log/nginx/test.json.access.log custom_json;
     access_log /var/log/nginx/test.access.log custom;
     error_log /var/log/nginx/test.error.log error;
     root /usr/share/nginx/html;
   }
   ```

   Note: Closely inspect the file contents if you used the copy-and-paste function. You should indent the lines as shown in the example, additionally there may be other formatting issues to fix.

3. When done adding the content to the file, quit the editor by first pressing the **ESC** key and then typing:

```
:wq
```

4. Reload your NGINX configuration with

```
$ nginx -s reload
```

5. Install the **epel** tools with:

```
$ yum install epel-release -y
```

```
Installed:
  epel-release.noarch 0:7-11

Complete!
```

The **epel-release** should show an installed message, as above.

6. Install the jq tool with:

```
$ yum install jq -y
```

```
Installed:
  jq.x86_64 0:1.6-2el7

Dependency Installed:
  oniguruma.x86_64 0:6.8.2-1.el7

Complete!
```

The **jq** tool and it's dependency should show an installed message, as above.

7. Generate some logging entries with:

```
$ curl http://localhost/test?1234
```

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.17.6</center>
</body>
</html>
```

As expected, the response to this query shows an error occurred.

20. View the access log to see the details of the unsuccessful connection:

```
$ tail /var/log/nginx/test.json.access.log | jq
```

```
  "request": "GET /test?1234 HTTP/1.1",
  "status": "404",
  "client": "127.0.0.1",
  "proxy": "127.0.0.1",
  "upstream": ""
}
```

You can see the connection message for the most recent request appears in the JSON custom format configured in the above steps. You will find a matching request also appears in the error log.

End of Lab