

# NGINX Core Lab Guide

## STUDENT LAB GUIDE

## Table of Contents

<b>Getting Started .....</b>	<b>2</b>
<b>Lab 1: Getting Started with NGINX Plus .....</b>	<b>3</b>
<b>Lab 2: View NGINX Processes .....</b>	<b>6</b>
<b>Lab 3: Serving Pages .....</b>	<b>9</b>
<b>Lab 4: Server Selection.....</b>	<b>11</b>
<b>Lab 5: Serving Content: Location.....</b>	<b>13</b>
<b>Lab 6: Using Variables .....</b>	<b>15</b>
<b>Lab 7: Logging.....</b>	<b>17</b>
<b>Lab 8: Proxying HTTP Requests .....</b>	<b>20</b>
<b>Lab 9: Routing HTTP Requests .....</b>	<b>22</b>
<b>Lab 10: Mapping Variables.....</b>	<b>28</b>
<b>Lab 11: Load Balancing.....</b>	<b>31</b>
<b>Lab 12: Using the NGINX API .....</b>	<b>35</b>
<b>Lab 13: Health Checks.....</b>	<b>44</b>
<b>Lab 14: Caching.....</b>	<b>46</b>
<b>Lab 15: Encrypting Web Traffic.....</b>	<b>48</b>



## Getting Started

This lab guide provides step-by-step instructions for lab exercises. Each lab corresponds to a module covered in class and provides you with hands-on experience working with the NGINX Controller software.

### Course Pre-requisites

This course is intended to introduce IT operations administrators to using the NGINX Plus platform. It is assumed students have some familiarity with:

- IT operations
- Web servers
- Linux
- Text editor: Vim, Vi, Emacs, etc.
- Networking

### Log into Hosted Environment

- Open your email and find the machine assigned to you. We are using Amazon Web Services' virtual private cloud.
- There is a login for each machine with a user name (student<#> and password). Please wait for instructions before logging into your machine.



## Lab 1: Getting Started with NGINX Plus

### Learning Objectives

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

- Use Secure Shell to log into your AWS machine instances

### Lab Exercises

Exercise 1: Log into your AWS instance

Bash Commands

Command	Description
ssh <username> @<FQDN>	Log into AWS instance
sudo	Enter prior to commands to elevate privileges to root user (super user do)



---

## Exercise 1: Log into your AWS Instance

---

### Overview

In this exercise, you log into your AWS Ubuntu instance. NGINX Plus is already installed on it.

### Steps

1. Open your email and find the machine assigned to you. It has a FQDN that looks like this: `ec2-13-56-78-94.us-west-1.compute.amazonaws.com`.

We have provided a login for the machine (username = student<#>, and password is listed after student name).

Please keep this machine name and password available throughout the class.

2. Log into the machine using SSH:
  - a. On a Mac laptop from a Terminal window, type the command

```
$ ssh student<#>@<ec2_host_name>
```

**Example:** `ssh student4@ec2-54-219-170-12.us-west-1.compute.amazonaws.com`

- b. On a Windows laptop, use an SSH client like Putty to log into a terminal window, using the command above. If you do not have an SSH client, download PuTTY from <https://www.putty.org>

---

## Exercise 2: Test the NGINX Instance in Your Browser

---

### Overview

In this exercise, you test the NGINX Plus instance in a browser.



## Steps

1. Paste the FQDN of your AWS instance in your browser. You should see the following response.



## Lab 2: View NGINX Processes

### Learning Objectives

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

- Determine what NGINX processes are running on Ubuntu
- View the master nginx configuration file

### Lab Exercises

Exercise 1: View NGINX processes and the configuration file

---

#### Exercise 1: Viewing Configurations

---

### Learning Objectives

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

- Determine the nginx processes running
- Determine the process ID location
- Use basic NGINX commands

### Overview

In this exercise, you check the main NGINX configuration file and processes.

### Steps

1. You should already be logged into your Ubuntu instance. If your system is non-responsive, log into it again, using the username and password given in your email.

```
$ ssh student<#>@<ec2-instance>
```

2. Determine the NGINX version by using the following command:

```
$ nginx -v
```

You see two versions – the open source on which the NGINX Plus version is built, and the NGINX Plus version.



- Determine what NGINX processes are running with the following Unix command:

```
$ ps aux | grep nginx
```

You should see that there are two NGINX processes running (master and worker) and the master process ID (PID) is listed.

```
$ ps aux | grep nginx
root      1264    0.0  0.1  32784   860 ?        Ss   Jun05   0:00 nginx: master process
nginx     1265    0.0  0.5  33184  3104 ?        S    Jun05   0:00 nginx: worker process
```

- Verify the process ID by using the following command:

```
$ cat /var/run/nginx.pid
```

```
$ cat /var/run/nginx.pid
1264
```

This indicates that the master process ID is indicated in the .pid file.

- Change directories to the master configuration file directory:

```
$ cd /etc/nginx
```

- List the files in the directory:

```
$ ls -F
```

You should see the `nginx.conf` file and the `conf.d` directory ("/" indicates a directory).

- Next, we'll check to make sure the configuration file (`nginx.conf`) has correct syntax (we're required to use the super user command to do most of the commands as we are not logged on as root user):

```
$ sudo nginx -t
```

You'll receive a message that the syntax is ok.

- To display all of the current configurations, use the following command:

```
$ sudo nginx -T
```





This concatenates all the NGINX configuration files and displays them in the terminal.

9. Now we'll just view the main `nginx.conf` file:

```
$ cat nginx.conf
```

Find the following sections in the `nginx.conf` file: `events`, `http`, and `stream`. How are these configured?



## Lab 3: Serving Pages

### Learning Objectives

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

- Create a backup to the default.conf file
- Create and test a new configuration file

### Lab Exercises

Exercise 1: Update a configuration file

---

#### Exercise 1: Working with Configuration Files

---

### Learning Objectives

- Create and test a new configuration file

### Overview

In this exercise, find and replace the included default configuration file.

### Steps

1. View the default.conf file
  - a. Change the directory:  

```
$ cd /etc/nginx/conf.d
```
  - b. List the file in your terminal:  

```
$ cat default.conf
```
  - c. What is NGINX listening for?
  - d. Where is it finding the main Web page?



2. Back up the default.conf file:

```
$ sudo mv default.conf default.conf.bak
```

3. Use the vim editor to create a new configuration file called server1.conf:

```
$ sudo vim server1.conf
```

4. Type "a" to enter editing mode.

5. Add the following to your file:

```
server {  
    listen 80;  
    root /home/ubuntu/public_html;  
}
```

6. Save the file:

Press the **esc** key to exit editing mode, then type: **:wq** to save and exit the file with changes (write, quit). Note: to save and exit a file without changes, use the **esc** key, then **:q!**

7. Save and reload NGINX:

```
$ sudo nginx -s reload  
(or $ sudo nginx -t && sudo service nginx reload)
```

8. Test your host using cURL:

```
$ curl http://localhost/
```

9. Test your host in a browser:

```
http://<ec2-hostname>/
```

10. Look at the list of files in the /home/ubuntu/public\_html directory:

```
$ ls /home/ubuntu/public_html
```

11. Look at the index.html file:

```
$ cat /home/ubuntu/public_html/index.html
```

12. This is the page you just saw. How does NGINX know to serve this Web page?



## Lab 4: Server Selection

### Learning Objectives

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

- Create and test a new configuration file
- Determine which listen directive will serve a request

### Lab Exercises

Exercise 1: Update a configuration file and test it.

---

#### Exercise 1: Server Selection

---

### Learning Objectives

- Create and test a new configuration file

### Overview

In this exercise, back up the current `server1.conf` configuration file. Create a new configuration file that uses two server blocks with different listen directives and determine which server responds.

### Steps

1. Back up the `server1.conf` file:

```
$ sudo mv server1.conf server1.old
```

2. Create a `server_test.conf` file and create two server test blocks as shown here:

```
server {  
    listen 80;  
    return 200 "this server listens on 0.0.0.0:80\n";  
}  
server {  
    listen 127.0.0.1:80;  
    return 200 "this server listens on 127.0.0.1:80\n";  
}
```



```
}
```

3. Save and exit the `server_test.conf` file (`esc, :wq`).

4. Reload NGINX:

```
$ sudo nginx -s reload
```

5. Before you test the configuration, predict which server will respond and why:

```
$ curl http://localhost
```

6. Which server responded? Why?



## Lab 5: Serving Content: Location

### Learning Objectives

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

- Create and test a new configuration file
- Determine which location block will serve a request

### Lab Exercises

Exercise 1: Update a configuration file and test it.

---

#### Exercise 1: Location Blocks

---

### Learning Objectives

- Create and test a new configuration file

### Overview

In this exercise, find and replace the included default configuration file. Use two server blocks with different listen directives and determine which server responds.

### Steps

1. Back up the `server_test.conf` file:

```
$ sudo mv server_test.conf server_test.bak
```

2. Rename the `server1.old` file to make it active:

```
$ sudo mv server1.old server1.conf
```

3. Open the `server1.conf` file and add code to create 3 locations. You are creating two locations that have content already in them, and adding a root directive to the `/images` location:



```
server {
    listen 80;
    root /home/ubuntu/public_html;

    location /application1 {
    }

    location /application2 {
    }

    location /images {
        root /data;
    }
}
```

4. Save the file and reload NGINX (**\$ sudo nginx -s reload**).
5. Test the locations in a browser as follows (test all locations before you take the next step):

```
http://<ec2-name>
http://<ec2-name>/application1
http://<ec2-name>/application2
http://<ec2-name>/images/logo.png
```

6. Why do you think you received a “403 Forbidden” on the application URLs?
7. List the files in the application1 directory:

```
$ ls /home/ubuntu/public_html/application1
```

8. Note there is no `index.html` file, which NGINX looks for by default. Update your `server1.conf` file by adding the correct index file:

```
location /application1 {
    index app1.html;
}

location /application2 {
    index app2.html;
}
```

9. Reload NGINX and retest the `application1` and `application2` endpoints in your browser.



## Lab 6: Using Variables

### Learning Objectives

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

- Create a configuration file with variables
- Determine effective behavior of variables within different scopes

### Lab Exercises

Exercise 1: Create a `variable_test` configuration file and test it.

---

#### Exercise 1: Setting Variables

---

### Learning Objectives

- Create and test a new configuration file.

### Overview

In this exercise, create a server block with three locations to test variable configurations and behavior.

### Steps

1. Rename your `server1.conf` file to `server1.old`:

```
$ sudo mv server1.conf server1.old
```

2. Create a new configuration file as follows:

```
$ sudo vim variable_test.conf
```

3. Add the following configuration:





```

server {
    listen 80;
    root /home/ubuntu/public_html;

    location / {
        return 200 "custom variable is $custom_variable
\n";
    }

    location /test1 {
        set $custom_variable 42;
        return 200 "custom variable is $custom_variable
\n";
    }

    location /test2 {
        set $custom_variable "null";
        return 200 "custom variable is $custom_variable
\n";
    }
}

```

4. Save the file and reload NGINX.
5. Test your configuration in the terminal with the following commands:

```

$ curl http://localhost/test1
$ curl http://localhost/test2
$ curl http://localhost/test

```

You don't receive a response for the /test location because a variable has not been set in that context.

6. Set `$custom_variable` in the server location:

```

server {
    listen 80;
    root /home/ubuntu/public_html;
    set $custom_variable 0;
    ...
}

```

7. Save the file and reload NGINX.
8. Test the same endpoints. What changed? What did not? Why?



## Lab 7: Logging

### Learning Objectives

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

- Set up an error and access log
- Create and test a custom log

### Lab Exercises

Exercise 1: Add logging directives to your configuration file and test them.

Exercise 2: Customize the access log with additional variables.

---

#### Exercise 1: Setting up Logging

---

### Learning Objectives

- Create and test a new configuration for logging

### Overview

In this exercise, add an error log and an access log configuration in the server block.

### Steps

1. Rename your `variable_test.conf` file to `variable_test.old`.

```
$ sudo mv variable_test.conf variable_test.old
```

2. Rename your `server1.old` file to `server1.conf`

```
$ sudo mv server1.old server1.conf
```

3. Open your `server1.conf` file and update it with the following changes.
  - a. Add an error log directive in the server context with a level of `info`. Add an access log directive in the server context with a type/name of `"combined"`.



- b. Save the file and reload NGINX.

Your file should read as follows:

```
server {  
    error_log /var/log/nginx/server1.error.log info;  
    access_log /var/log/nginx/server1.access.log combined;  
  
    listen 80;  
    root /home/ubuntu/public_html;  
  
    location /application1 {  
        index appl.html;  
    }  
  
    location /application2 {  
        index app2.html;  
    }  
  
    location /images {  
        root /data;  
    }  
}
```

4. Save the file and reload nginx.
5. Open a browser and test your instance: `http://<ec2-host>/application1`
6. Look at the access log using the following command:

```
$ cat /var/log/nginx/server1.access.log
```

---

### Exercise 2: Create a Custom Log

---

## Learning Objectives

- Create and test a custom configuration for logging

## Overview



In this exercise, change the format of your access log by adding a log format directive and referencing its custom name/type.

## Steps

1. Add the following `log_format` command to your `server1.conf` file in the `http` context (i.e. at the top of the file).

```
log_format test_log '"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 Proxy_Hostname: $proxy_host\n Real_IP: $http_x_real_ip"';

server {
    error_log /var/log/nginx/server1.error.log info;
    access_log /var/log/nginx/server1.access.log test_log;

    listen 80;
    root /home/ubuntu/public_html;
    . . .
}
```

2. Be sure to change the “combined” access log type to match the log format command (`test_log`).
3. Save the file and reload NGINX.
4. Send another request to your instance: `http://<ec2-host>/application1`
5. Look at the access log using the following command:

```
$ cat /var/log/nginx/server1.access.log
```

You should see that the logfile is now providing one variable per line with a descriptive indicator before each.



## Lab 8: Proxying HTTP Requests

### Learning Objectives

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

- Set up a back-end server
- Create and test a proxy to the back-end server

### Lab Exercises

Exercise 1: Create second configuration file for the back-end server and proxy to it

---

#### Exercise 1: Proxying HTTP Requests

---

### Learning Objectives

- Create and test the `proxy_pass` directive

### Overview

In this exercise, add a second server configuration file that listens on a different port than port 80. Point to that server from the first configuration file using the `proxy_pass` directive.

### Steps

1. Create a new configuration file called `server2.conf` using this command:

```
$ sudo vim /etc/nginx/conf.d/server2.conf
```

2. Create the following server block, which listens on port 8080 and sends you to the `index.html` file in the `/data/server2` directory.

```
server {  
    listen 8080;  
    root /data/server2;  
}
```

3. Save the file.



4. View the contents of the index.html file that you'll be proxying to:

```
$ cat /data/server2/sampleApp/index.html
```

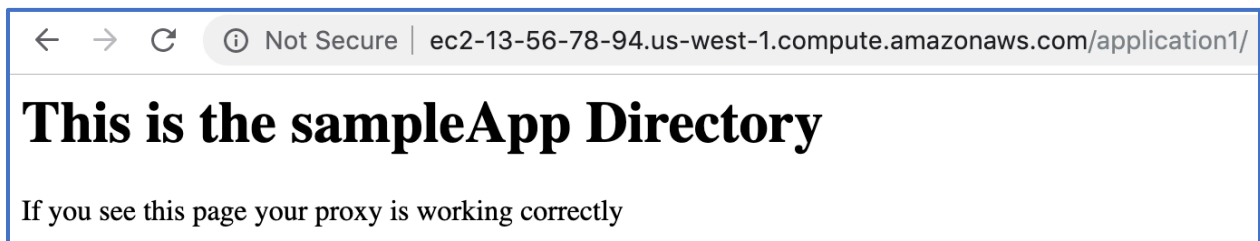
5. Open server1.conf to set up your proxy. Add the following proxy\_pass statement under your /application1 location:

```
location /application1 {  
    index appl.html;  
    proxy_pass http://localhost:8080/sampleApp;  
}
```

6. Save the file and reload NGINX.
7. Test the proxy statement in a browser (refresh the browser if necessary):

```
http://<ec2-host>/application1
```

8. You should see the following:



## Lab 9: Routing HTTP Requests

### Learning Objectives

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

- Use NGINX directives to reroute traffic
- Define URL rewrites
- Determine rewrite request processing

### Lab Exercises

Exercise 1: Use the `alias` directive

Exercise 2: Set up and test a rewrite URL

Exercise 3: Set up a rewrite with and without the `break` flag

---

#### Exercise 1: Using the `alias` Directive

---

### Learning Objectives

- Use the `alias` directive with a location defined by a regular expression.

### Overview

You set up a location using a regular expression that captures incoming HTTP requests looking for the `/pictures` URI, followed by any character before a dot, followed by either an extension of `gif`, `jpe`, `jpg`, or `png`. In other words, the location block serves URIs that have `/pictures/<file_name>.<picture_file_extension>` in them. You also specify a replacement path using the `alias` directive.

### Steps

1. Open the `server1.conf` configuration file:

```
$ sudo vim /etc/nginx/conf.d/server1.conf
```



2. Comment out the /images location block.

```
#location /images {  
    #root/data;  
#}
```

3. Create a new location using the following case insensitive regular expression and alias directive and place in the file. For example:

```
listen 80;  
root /home/ubuntu/public_html;  
  
location ~ ^/pictures/(.+\. (gif|jpe?g|png))$ {  
    alias /data/images/$1;  
}  
  
location /application1 {  
    index appl.html;  
    proxy_pass http://localhost:8080/sampleApp;  
}
```

4. Save the file and reload NGINX.
5. In a browser, test **http://<ec2-host>/pictures/logo.png**

---

## Exercise 2: Setting Up Rewrite Data

---

### Learning Objectives

- Use the `rewrite` directive with a regular expression to redirect traffic.

### Overview

You set up a `rewrite` directive to capture any URI request to `/shop/greatproducts/<#>` and send it to `/shop/product/product/<#>`. You add your host name to the `product1.html` page to reroute that request to the logo picture to the `/media/pics` directory. This won't work, but we'll fix it in the next lab.





## Steps

1. Check the location for the product<#>.html files:

```
$ ls /home/ubuntu/public_html/shop/product
```

2. Open the product1.html file:

```
$ sudo vim  
/home/ubuntu/public_html/shop/product/product1.html
```

3. Edit the <p> tag. Replace <ec2-hostname> with your ec2 instance. For example:

```
<h1>This is product 1</h1>  
<p>  
  
</p>
```

4. Save the file.

5. Open server1.conf and define a rewrite in the server context with the following regular expression and replacement string:

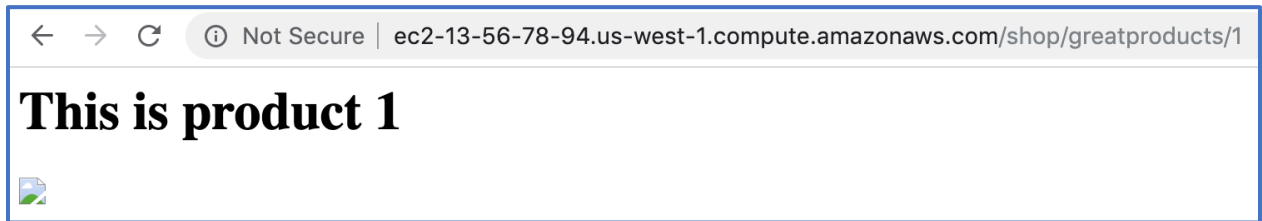
```
server {  
    error_log /var/log/nginx/server1.error.log info;  
    access_log /var/log/nginx/server1.access.log test_log;  
  
    rewrite ^/shop/greatproducts/(\d+)$ /shop/product/product$1.html;  
  
    listen 80;
```

6. Save the file and reload NGINX.
7. Test the following URLs in your browser:

```
http://<ec2-host>/shop/greatproducts/2  
http://<ec2-host>/shop/greatproducts/3  
http://<ec2-host>/shop/greatproducts/1
```



You receive text messages for `product2.html` and `product3.html`. The `product1.html` page serves a broken image link because the image is not in the location we're pointing to in the `server1.conf` file (`/media/pics`).



8. Now we'll add another rewrite in the server context (below the first one) to correct this. This rewrites the `media/pics` directory to the `/pictures` directory:

```
rewrite ^/media/pics/(.+\. (gif|jpe?g|png))$ /pictures/$1;
```

9. Save the file and reload NGINX.
10. Re-test the `/shop/greatproducts/1` request. The image should appear below the text.



---

### Exercise 3: Setting Up Rewrite Flags

---

## Learning Objectives

- Set up a rewrite directive with and without the break flag.

## Overview

You move the `shop/greatproducts` rewrite from the server context into a new `location /shop` context. You also create an additional rewrite in the same `/shop` context, as well as a `return` directive, and test these with and without break flags.

## Steps

1. Open `server1.conf` and comment out the `shop/greatproducts` rewrite. Add a `location /shop` and copy the `shop/greatproducts` rewrite directive to that location. Add an additional rewrite as shown below, as well as a `return 403` directive to the location block. Here is your finished example:

```
server {
    error_log /var/log/nginx/server1.error.log info;
    access_log /var/log/nginx/server1.access.log test_log;

#    rewrite ^/shop/greatproducts/(\d+)$ /shop/product/product$1.html;
    rewrite ^/media/pics/(.+\.gif|jpe?g|png))$ /pictures/$1;

    location /shop {
        rewrite ^/shop/greatproducts/(\d+)$ /shop/product/product$1.html;
        rewrite ^/shop/./(\d+)$ /shop/services/service$1.html;
        return 403;
    }

    listen 80;
    root /home/ubuntu/public_html;
```

2. Save the file and reload NGINX.
3. Test the `/shop/greatproducts/1` URL in a browser.

You receive a 403 forbidden error. Why?



Because there are no break flags, NGINX continues to process all the rewrites in order, and then processes the `return 403` directive.

4. Correct this issue by placing the break flag at the end of each rewrite in your `location /shop` block:

```
location /shop {  
    rewrite ^/shop/greatproducts/(\d+)$ /shop/product/product$1.html break;  
    rewrite ^/shop/.+/(d+)$ /shop/services/service$1.html break;  
    return 403;  
}
```

5. Save the file and reload NGINX.
6. Test the `/shop/greatproducts` URL again. You receive the `product1.html` Web page again.



## Lab 10: Mapping Variables

### Learning Objectives

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

- Use the map directive with a regular expression to determine a location
- Use the map directive with an `if` statement to set up conditional logging

### Lab Exercises

Exercise 1: Use a map to determine location

Exercise 2: Use a map to set up conditional logging

---

#### Exercise 1: Using the map Directive

---

### Steps

1. In the `server1.conf` file, create the following map in the http context (place it at the very top of the file, above the server context):

```
map $uri $is_redirect {
    default      0;
    /test1       1;
    /test2       2;
    /test3       3;
}
```

2. In the server context, add a regex location for `/test`, followed by any digit as follows:

```
server {
    . . . .
    location ~* /test(\d+)$ {
        return 200 "variable = $is_redirect \n";
    }
    . . . .
}
```

3. Save the file and reload NGINX.



4. Test the map as follows:

```
$ curl -k http://localhost/test1
```

5. Try testing other values:

```
$ curl -k http://localhost/test2
$ curl -k http://localhost/test3
$ curl -k http://localhost/test4
$ curl -k http://localhost/test4241234
```

---

## Exercise 2: Map Directive with Conditional Logging

---

### Learning Objectives

- Use the map directive with an `if` statement to set up conditional logging.

### Overview

You add a map with the `$loggable` variable to log access when the returned status code starts with a number other than 2 or 3.

### Steps

1. In the `server1.conf` file, in the `http` context (top of file), create a map for excluding response codes that begin with a 2 or a 3, that is 200 or 300:

```
map $status $loggable {
    ~^[23]    0;
    default   1;
}
```

2. In the `access_log` directive, add the `if` parameter with the new custom variable (`$loggable`) as follows:

```
access_log /var/log/nginx/server1.access.log test_log if=$loggable;
```

3. Save the file and reload NGINX.



4. Make a request for:

`$ curl http://localhost`, then one that doesn't exist, such as:

`$ curl http://localhost/nowhere/`

5. View the access log:

`$ sudo cat /var/log/nginx/server1.access.log`

You should see that only a status 404 – 500 codes appear.

```
'"Request:GET /nowhere HTTP/1.1
Status:404
Request_URI:/nowhere
Host:localhost
Client_IP:127.0.0.1
Proxy_IP(s):127.0.0.1
Proxy_Hostname:-
Real_IP:-"'
```



## Lab 11: Load Balancing

### Learning Objectives

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

- Set up and test load balancing methods

### Lab Exercises

Exercise 1: Set up and test the round robin method of load balancing with and without weights

Exercise 2: Set up the hash method of load balancing

---

#### Exercise 1: Round Robin Load Balancing

---

### Learning Objectives

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

- Set up and test round robin load balancing
- Add and test weights on round robin load balancing

### Overview

In this exercise, you set up three back end servers with round robin load balancing. You test the configuration with and without weights.

### Steps

1. Create the new configuration file, `backends.conf` in the `/etc/nginx/conf.d` directory. Add three virtual servers with the following root directives and ports:

```
server {
    listen 8081;
    root /data/backend1;
}

server {
    listen 8082;
    root /data/backend2;
}
```





```
server {
    listen 8083;
    root /data/backend3;
}
```

2. Save the file and reload NGINX.

3. Test the back ends in your terminal:

```
$ curl http://localhost:8081
$ curl http://localhost:8082
$ curl http://localhost:8083
```

4. Back up server1.conf and server2.conf files:

```
$ sudo mv server1.conf server1.conf.bak
$ sudo mv server2.conf server2.conf.bak
```

5. Create a new configuration file called myServers.conf with an upstream context that points to each new server:

```
upstream myServers {
    server 127.0.0.1:8081;
    server 127.0.0.1:8082;
    server 127.0.0.1:8083;
}
```

6. In the same file, create a server listening on port 80 with the following root and log directives. Create a location that proxies all requests to the myServers back end (server pool):

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

    location / {
        proxy_pass http://myServers;
    }
}
```

7. Save the file and reload NGINX.



8. Test the load balancer configuration by curling several times:

```
$ curl http://localhost/
```

You should see NGINX cycle through the three web servers, one after the other as follows:

```
$ curl http://localhost/
<h1>This is Backend1</h1>
$ curl http://localhost/
<h1>This is Backend2</h1>
$ curl http://localhost/
<h1>This is Backend3</h1>
```

9. Now open the `myServers.conf` file and add a weight to the second server:

```
upstream myServers {
    server 127.0.0.1:8081;
    server 127.0.0.1:8082 weight=2;
    server 127.0.0.1:8083;
}
```

10. Save the file and reload NGINX
11. Test the configuration. You see server 2 serving twice the number of requests as before.

---

### Exercise 2: Hash Method Load Balancing

---

## Learning Objectives

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

- Set up and test the hash method of load balancing



## Overview

In this exercise, you remove the weights on your second server and change the load balancing method from the default round robin to the hash method. You use the request URI as the hash key, so changing the request URI should move you to the next server but keeping the same request URI should keep you on the same server.

## Steps

1. Open the `myServers.conf` file and add the following hash directive in the upstream context. Remove the weight from the second virtual server:

```
upstream myServers {  
    hash $scheme$host$request_uri;  
    server 127.0.0.1:8081;  
    server 127.0.0.1:8082;  
    server 127.0.0.1:8083;  
}
```

2. Save the file and reload NGINX.
3. Test the configuration by curling localhost with various arguments:

```
$ curl http://localhost/?1  
$ curl http://localhost/?2
```

You should see results similar to these:

```
$ curl http://localhost/?1  
<h1>This is Backend1</h1>  
$ curl http://localhost/?1  
<h1>This is Backend1</h1>  
$ curl http://localhost/?2  
<h1>This is Backend3</h1>  
$ curl http://localhost/?2  
<h1>This is Backend3</h1>  
$ curl http://localhost/?1  
<h1>This is Backend1</h1>
```



## Lab 12: Using the NGINX API

### Learning Objectives

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

- Configure NGINX Plus to view performance metrics on the dashboard
- Use the NGINX Plus API to configure back end servers

### Lab Exercises

Exercise 1: Define and test a dashboard page

Exercise 2: Specify and test a server zone for each server

Exercise 3: Enable the dynamic API and execute API commands

Exercise 4: Enable and test back end persistent state changes

---

#### Exercise 1: Defining a Dashboard

---

### Learning Objectives

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

- Set up and test a dashboard page

### Overview

In this exercise, you create a location for the NGINX Plus api and set up a shared memory zone in the upstream group to allow view (GET) access to your server pool (upstream group).

### Steps

1. In the `myServers.conf` file, in the server context, define a location for the api and a location for the dashboard as follows:
  - a. Comment out the hash load balancing directive:

```
# hash $scheme$host$request_uri;
```



- b. Enter a shared memory zone at the top of the upstream block:

```
upstream myServers {  
    #hash $scheme$host$request_uri;  
    zone http_backend 64k;  
    server 127.0.0.1:8081;  
    . . . .
```

- c. Add a new server block on port 8080 with two location blocks - one for the api and one for the dashboard:

```
server {  
    listen 8080;  
    root /usr/share/nginx/html;  
  
    location /api {  
        api;  
    }  
  
    location /dashboard.html {  
    }  
}
```

2. Save the file and reload NGINX.

3. Test in a browser:

`http://<ec2-host>:8080/dashboard.html`

You should see your dashboard as follows:



Upstreams
 Shared zones

[nginx-plus-r18 \(1.15.10\)](#)  
 Address **172.31.29.234**  
 PID **1264**  
 Uptime **2m**

**Connections** SSL Accepted:866

Current	Accepted/s	Active	Idle	Dropped
6	0	1	5	0

**Requests** Total:3440

Current	Req/s
1	7

**Upstreams**

Total Problems  
**1 / 0**

**Servers**  
 All: 3 / Up: 3  
 Failed: 0

4. Click the Upstreams tab to view the server information:

Upstreams
 Shared zones

Upstreams [Show upstreams list](#)

Failed only ☐

**myServers** Zone: **40 %** [Show all](#)

Server		Requests			Responses		Conns		Traffic				Server checks		Health monitors			Response time			
Name	DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers	Response
127.0.0.1:8081	0ms	1	0	0		0	0	0	∞	0	0	0	0	0	0	0	0	0	-	-	-
127.0.0.1:8082	0ms	1	0	0		0	0	0	∞	0	0	0	0	0	0	0	0	0	-	-	-
127.0.0.1:8083	0ms	1	0	0		0	0	0	∞	0	0	0	0	0	0	0	0	0	-	-	-

5. Click the Shared zones tab to view the name of your shared memory zone and memory usage:

Upstreams
 Shared zones

Shared zones

Zone	Total memory pages	Used memory pages	Memory usage
http_backend	15	6	40 %



---

## Exercise 2: Create Server Memory Zones

---

### Learning Objectives

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

- Set up and test a dashboard page for individual or grouped server metrics

### Overview

In this exercise, you configure a shared memory zone for each server to allow view access of your server metrics.

### Steps

1. In your `backends.conf` file, add a shared memory zone for each server using the `status_zone` directive. You can use the same zone for any servers you would like to group together for metric information. Here is one example:

```
server {
    listen 8081;
    root /data/backend1;
    status_zone USA;
}

server {
    listen 8082;
    root /data/backend2;
    status_zone USA;
}

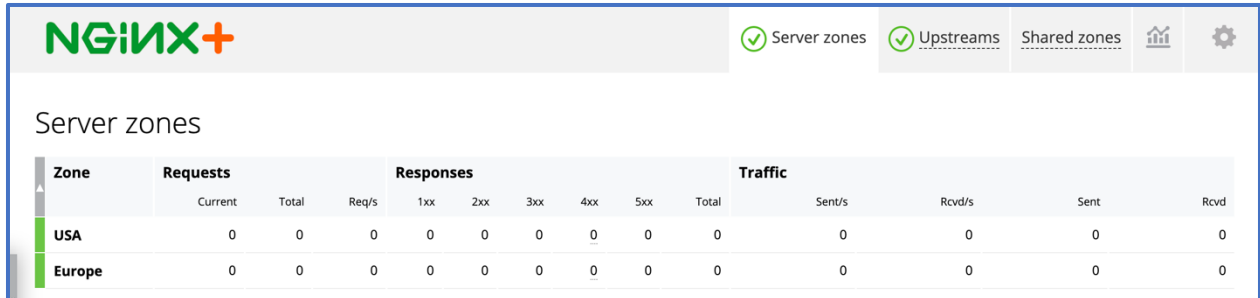
server {
    listen 8083;
    root /data/backend3;
    status_zone Europe;
}
```

2. Save the file and reload NGINX.
3. Test the dashboard page:

`http://<ec2-host>:8080/dashboard.html`



You have a new tab called “HTTP zones”, where you can view the server zone metrics:



Zone	Requests		Responses							Traffic			
	Current	Total	Req/s	1xx	2xx	3xx	4xx	5xx	Total	Sent/s	Rcvd/s	Sent	Rcvd
USA	0	0	0	0	0	0	0	0	0	0	0	0	0
Europe	0	0	0	0	0	0	0	0	0	0	0	0	0

---

### Exercise 3: Enable Configuration with API

---

## Learning Objectives

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

- Enable access to the API in order to write configuration changes
- Execute API commands to configure backend servers

## Overview

In this exercise, you enable the api location with write access and use cURL commands to remove a server from the backend server pool. You add the server back in with a different weight, and you view these changes in your dashboard.

## Steps

1. Enable the api location to allow write access. In the `myServers.conf` file, add the `write=on` parameter to the `api` directive. This enables you to send api commands to your configuration (PUT, POST, PATCH, and DELETE).

```
location /api {  
    api write=on;  
}
```

2. Save the file and reload NGINX.





- Click the Upstreams tab of your NGINX dashboard page. You see 3 servers there. Watch this page as you Send the following API command using cURL:

```
$ curl -X DELETE
"http://localhost:8080/api/3/http/upstreams/myServers/servers/2" -H "accept: application/json" | jq
```

This deletes your second server and pipes the output/result to your terminal in jQuery so it's easy to read. You also see your dashboard change to show only the first two servers.

- Reload NGINX in the terminal and continue to watch your dashboard page. Why does the third server re-appear?
- Use the same curl command as in step 3 to remove your server. Then add it back with a weight of 5:

```
$ curl -X POST
"http://localhost:8080/api/3/http/upstreams/myServers/servers/"
-H "accept: application/json" -H "Content-Type: application/json" -d "{ \"server\": \"127.0.0.1:8083\", \"weight\": \"5\"}" | jq
```

You'll see that the server has been added back with a new weight of 5:

## Upstreams

Show upstreams list

**myServers** Zone: 40 %

Server			Requests		Responses		
Name	DT	W	Total	Req/s	...	4xx	5xx
127.0.0.1:8081	0ms	1	0	0		0	0
127.0.0.1:8082	0ms	1	0	0		0	0
127.0.0.1:8083	0ms	5	0	0		0	0



---

## Exercise 4: Persist State with the API

---

### Learning Objectives

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

- Use the state directive to allow the API changes to be permanent

### Overview

In this exercise, you create a directory for the state file and provide write permissions to NGINX on that file. You remove the servers from the upstream block in your configuration file and add the `state` directive. Finally, you use the API to re-create the upstream servers and test to determine that the changes persist with this configuration.

### Steps

1. Create a directory for the state file:

```
$ sudo mkdir -p /var/lib/nginx/state
```

2. Change the ownership on the state file so that NGINX has write permissions to it:

```
$ sudo chown nginx:nginx /var/lib/nginx/state
```


3. Open the `myServers.conf` file and add the `state` directive as shown. Comment out the `server` directives in the upstream:

```
upstream myServer {  
    . . .  
    zone http_backend 64k;  
    state /var/lib/nginx/state/http_backend.state;  
    # server 127.0.0.1:8081;  
    # server 127.0.0.1:8082;  
    # server 127.0.0.1:8083;  
}
```

4. Save the file and reload NGINX.
5. View the dashboard. The servers have been removed.



Upstreams [Show upstreams list](#) Failed only ☐

**myServers**  Zone: 27% [Show all](#)


Server		Requests		Responses		Conns		Traffic		Server checks		Health monitors			Response time						
Name	DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers	Response
No servers with 'all' state found in this upstream group.																					

- Use the dashboard GUI (see next page) or use the API to create the upstream servers with the following command (this adds one server):

```
$ curl -X POST
"http://localhost:8080/api/3/http/upstreams/myServers/servers/" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-d "{ \"server\": \"127.0.0.1:8081\", \"weight\": \"4\"}" \
| jq
```

To use the dashboard to add servers, click the edit (pencil) button (refresh the browser to see this).

Upstreams [Show upstreams list](#)

**myServers**  [Edit selected](#) [Add server](#) Zone: 40%

Server		Requests		Responses		Conns		Traffic			
Name	DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s
<input type="checkbox"/> <b>127.0.0.1:8081</b>	0ms	4	0	0	▶	0	0	0	∞	0	

Click *Add server*. Enter information to add the server:



Add server to "myServers"✕

Server address

127.0.0.1:8082

Server route

☐ Add as backup server

weight

max\_conns

max\_fails

fail\_timeout

slow\_start

service

Set state

☒ Up

☐ Down

☐ Drain

Add

Cancel

7. Be sure to add back all 3 servers. Reload NGINX and see that the changes persist this time.



## Lab 13: Health Checks

### Learning Objectives

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

- Configure passive health checks
- Configure active health checks based on match block conditions

### Lab Exercises

Exercise 1: Configure a match block for server health checks

---

#### Exercise 1: Configure a Health Check

---

### Learning Objectives

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

- Set up a match block of three checks against server health and test it

### Overview

In this exercise, you set up a list of three conditions for a health check in a match block and add the `health_check` directive to the `location` / prefix that references it.

### Steps

1. Open the `myServers.conf` file and add a match block in the `http` context as follows:

```
match health_conditions {  
    status 200;  
    header Content-Type = text/html;  
    body !~ maintenance;  
}
```

The conditions are that the status response code must be 200, the header must be a text or html document and the word “maintenance” must not appear in the body of the document. Note that the check for “maintenance” is case sensitive.



2. In the location / prefix, add the health\_check directive (below the proxy\_pass directive) and reference the match block. The uri parameter points to a test file on the backend server (/health/test.html).

```
server {
    listen 80;
    . . .
    location / {
        proxy_pass http://myServers;
        health_check match=health_conditions fails=2
        uri=/health/test.html;
    }
}
```

3. Save the file and reload NGINX.
4. Refresh the dashboard. It shows that a server is down because the word "maintenance" is in the test.html file we're checking.

Upstreams

Show upstreams list

Failed only

myServers

Zone: 40 %

Show all

Server		Requests				Responses				Conns				Traffic				Server checks		Health monitors				Response time	
Name	DT	W	Total	Req/s	...	4xx	5xx	A	L	Sent/s	Rcvd/s	Sent	Rcvd	Fails	Unavail	Checks	Fails	Unhealthy	Last	Headers	Response				
127.0.0.1:8081	0ms	4	0	0		0	0	0	∞	0	0	0	0	0	0	3	0	0	passed	-	-				
127.0.0.1:8082	0ms	1	0	0		0	0	0	∞	0	0	0	0	0	0	3	0	0	passed	-	-				
127.0.0.1:8083	11.17s	1	0	0		0	0	0	∞	0	0	0	0	0	0	3	3	1	failed	-	-				

5. Open the /data/backend3/health/test.html file and delete the word "maintenance" in the body of the file (or change the "m" to "M"):

```
$ sudo vim /data/backend3/health/test.html
```

6. Save the file and refresh the dashboard – the server will be back up again.



## Lab 14: Caching

### Learning Objectives

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

- Set up a reverse proxy cache for upstream servers

### Lab Exercises

Exercise 1: Set up a proxy cache and test it.

---

#### Exercise 1: Set up the Proxy Cache

---

### Learning Objectives

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

- Set up a proxy cache and test the cache

### Overview

In this exercise, you set up a cache path with a validation of 5 minutes and check the cache icon in the dashboard.

### Steps

1. Add a proxy cache path in the http context of your myServers.conf file:

```
proxy_cache_path /data/nginx/cache levels=1:2
keys_zone=upstream_cache:20m inactive=5m max_size=2G;
```

2. In the server (using port 80) context, set the proxy\_cache\_key to the \$scheme, \$host, and \$request\_uri:

```
proxy_cache_key $scheme$host$request_uri;
add_header X-Proxy-Cache $upstream_cache_status;
```



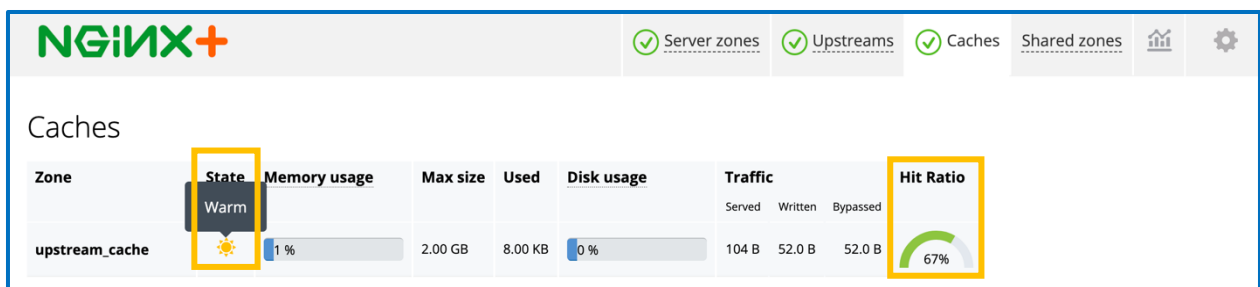
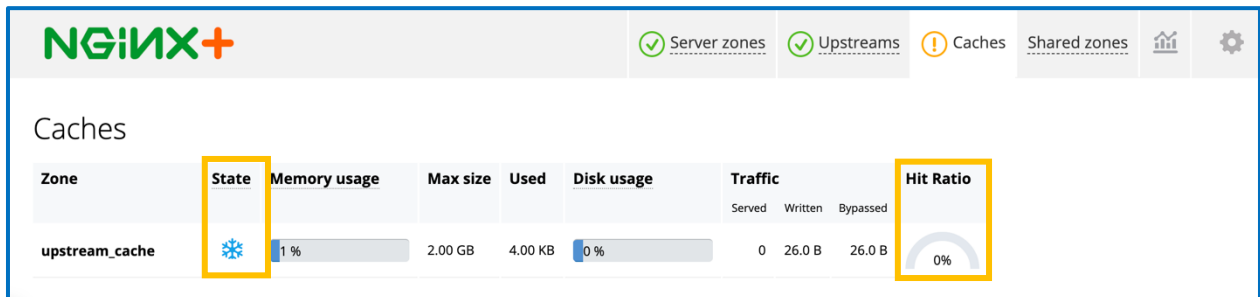
3. In the location / prefix, set the proxy\_cache as follows:

```
location / {  
    proxy_pass http://myServers;  
    health_check match=health_conditions fails=2  
    uri=/health/test.html;  
    proxy_cache upstream_cache;  
    proxy_cache_valid 200 5m;  
}
```

4. Save the file and reload NGINX.
5. Send several requests via the browser or use cURL:

```
$ curl -I http://localhost/
```

6. Refresh the dashboard on the Caches tab to see that the cache icon changes from cold to warm and that the hit ratio increases. (It may take a few moments for the state to change to warm.)





## Lab 15: Encrypting Web Traffic

### Learning Objectives

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

- Configure HTTPS access
- Set up encryption levels on the proxy server
- Test the security of your server against sslabs.com

### Lab Exercises

Exercise 1: Set up a directory for certificates and keys

Exercise 2: Configure the HTTPS access and test it

---

#### Exercise 1: Generate Certificate and Key

---

### Learning Objectives

By the end of the lab you will have:

- Set up the directory and run the `openssl` command to generate a self-signed certificate and key.

### Overview

In this exercise, you set up a directory and run the `openssl` command.

### Steps

1. Use the following command to create a directory for certificates and keys:

```
$ sudo mkdir /etc/nginx/ssl -p
```

2. Change directories to the ssl directory:

```
$ cd /etc/nginx/ssl
```

3. Run the `openssl` command and press return through all of the `openssl` prompts:

```
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:4096
```



```
-keyout nginx.key -out nginx.crt
```

---

## Exercise 2: Configure and Test SSL Parameters

---

### Learning Objectives

By the end of the lab you will have:

- Set up the ssl parameters according to NGINX best practices
- Test the configuration of the site against sslabs.com

### Overview

In this exercise, you set up the ssl parameters according to NGINX best practices and then force all traffic to https. Then you'll test the site on SSL Labs.com

### Steps

1. Create a file called `ssl-params.conf`:

```
$ sudo vim /etc/nginx/conf.d/ssl-params.conf
```

2. Paste the following configurations into the file:

```
ssl_certificate /etc/nginx/ssl/nginx.crt;  
ssl_certificate_key /etc/nginx/ssl/nginx.key;  
ssl_protocols TLSv1.2 TLSv1.3;  
ssl_ciphers "AES256+EECDH:AES256+EDH:!aNULL";  
ssl_prefer_server_ciphers on;  
ssl_session_cache shared:SSL:10m;  
ssl_session_timeout 10m;  
ssl_session_tickets off;  
add_header Strict-Transport-Security "max-age=63072000;  
includeSubdomains";  
add_header X-Frame-Options DENY;  
add_header X-Content-Type-Options nosniff;
```

3. Save the file.



4. Open the `myServers.conf` file and change the `listen` directive (port 80) to port 443 and add the `ssl` parameter:

```
server {  
    listen 443 ssl;  
    root /usr/share/nginx/html;  
    . . .  
}
```

5. Add a new server block above this one, forcing http traffic to https:

```
server {  
    listen 80;  
    return 301 https://$host$request_uri;  
}
```


6. Save the file and reload NGINX.
7. Test the access using the following cURL command (the `-i` parameter shows body content, the `l` parameter shows headers, the `L` parameter tells NGINX to follow any redirect, and the `k` parameter tells NGINX to ignore any SSL errors such as a self-signed certificate). You see both the return and the redirect:

**\$ curl -iILk http://localhost**

```
$ curl -iILk http://localhost/  
HTTP/1.1 301 Moved Permanently  
Server: nginx/1.15.10  
Date: Thu, 13 Jun 2019 18:56:14 GMT  
Content-Type: text/html  
Content-Length: 170  
Connection: keep-alive  
Location: https://localhost/  
Strict-Transport-Security: max-age=63072000; includeSubdomains  
X-Frame-Options: DENY  
X-Content-Type-Options: nosniff  
  
HTTP/1.1 200 OK  
Server: nginx/1.15.10  
Date: Thu, 13 Jun 2019 18:56:14 GMT  
Content-Type: text/html  
Content-Length: 26  
Connection: keep-alive  
Last-Modified: Mon, 08 Apr 2019 19:29:19 GMT  
ETag: "5caba10f-1a"  
Strict-Transport-Security: max-age=63072000; includeSubdomains  
X-Frame-Options: DENY  
X-Content-Type-Options: nosniff  
X-Proxy-Cache: HIT  
Accept-Ranges: bytes
```



8. Test the site on sslabs.com. When the mismatch message appears, click the link to ignore it.

 **Qualys. SSL Labs**

Home Projects Qualys Free Trial Contact

You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > ec2-18-144-21-195.us-west-1.compute.amazonaws.com

**SSL Report: ec2-18-144-21-195.us-west-1.compute.amazonaws.com** (18.144.21.195)

Assessed on: Thu, 13 Jun 2019 18:57:15 UTC | [Hide](#) | [Clear cache](#) [Scan Another »](#)

Certificate name mismatch


[Click here to ignore the mismatch and proceed with the tests](#)

Alternate names not found in the certificate

What does this mean?

We were able to retrieve a certificate for this site, but the domain names listed in it do not match the domain name you requested us to inspect. It's possible that:

9. Explore the Web page. What were the results?

 **Qualys. SSL Labs**

Home Projects Qualys Free Trial Contact


You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > ec2-18-144-21-195.us-west-1.compute.amazonaws.com

**SSL Report: ec2-18-144-21-195.us-west-1.compute.amazonaws.com** (18.144.21.195)

Assessed on: Thu, 13 Jun 2019 18:58:17 UTC | [Hide](#) | [Clear cache](#) [Scan Another »](#)

### Summary

Overall Rating



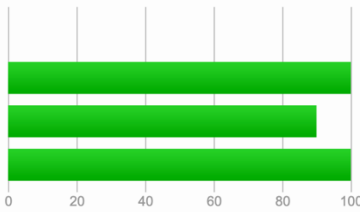
If trust issues are ignored: A

Certificate

Protocol Support

Key Exchange

Cipher Strength



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This server's certificate is not trusted, see [below](#) for details.

