F5 Training

Securing Applications with NGINX

Lab Only Guide

NGINX Plus R29

Publishing Version: 2023-06-16

Lab Only Guide **2023-06-16**

Support and Contact Information

Obtaining Technical Support

Web: support.f5.com (Ask F5)

Suggestions: feedback@f5.com

Contacting F5

Web: www.f5.com

Sales: sales@f5.com

Information: info@f5.com

Contact Training

F5 Information: www.f5.com/services/training

North America/South America: training@f5.com

Europe/Middle East/Africa: emeatraining@f5.com

Japan: japantraining@f5.com

Asia/Pacific: apactraining@f5.com

NGINX Information: www.nginx.com/learn/nginx-training

NGINX Training: NGINX-training@f5.com

Legal Notices

Trademarks

Materials and Patents

Table of Contents

Chapter 1

Labs - Introduction to Lab Environment

Lab Contents

1. **Lab Environment**
2. **Login to the NGINX training system**
3. **Test Web Application**

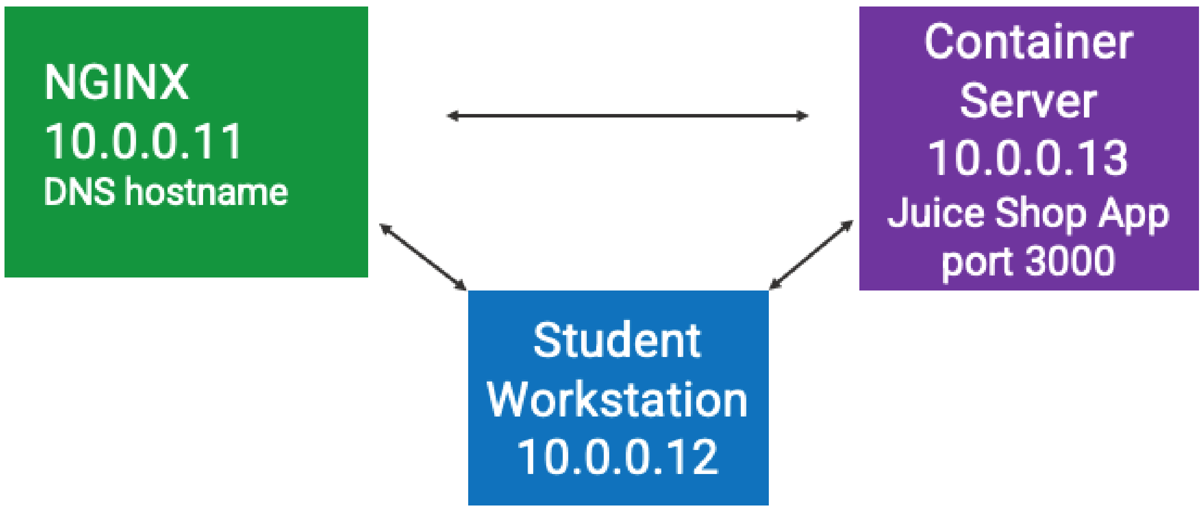
Estimated time to complete labs: **[XX] minutes**

Lab Credentials and IP Address List

LAB SYSTEM	**IP ADDRESS**
NGINX	10.0.0.11
Student Workstation	10.0.0.12
Container Server	10.0.0.13

There are 3 lab systems: NGINX, Student Workstation and Container Server. All lab systems use the username **student** with the password **student**.

Lab Network Topology



There are 3 lab systems: NGINX, Student Workstation and Container Server. The NGINX lab system has NGINX Plus installed. The Container Server system has the OWASP Juice Shop web application running on port 3000 using Kubernetes. The Student Workstation lab system is an Ubuntu Linux system that is a graphical system used to test access to the NGINX lab system and the Juice Shop application.

Lab 1: Login to Your Training Machine

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- The OWASP Juice Shop web application must be running on the Server Container lab system and the NGINX lab system must have a /etc/nginx/conf.d/juice.conf file specifying proxy_pass and upstream to where the Juice Shop app is running.

Overview

In this exercise, you practice logging into your NGINX lab system which is where most labs will be completed. Lab steps will explicitly say when to use the Student Workstation lab system instead of the NGINX lab system.

Objectives

At the end of this lab you will be able to:

- Login to your NGINX lab system
- Verify the version of NGINX installed
- Test access to the Juice Shop web application directly on the Server Container lab system
- Test access to the Juice Shop web application via the NGINX lab system
- View the NGINX configuration file for the Juice Shop web application

Steps

1. Log into your **NGINX** lab system. Use **student** as the *username* and as the *password*.

2. On the Desktop open the **Lab_Files** folder and then open the **SNIPPETS.txt** file. This file lists the commands and NGINX code used in the lab steps and can be used to copy and paste lines into the command line interface (terminal) or browser. It's highly recommend that you copy/paste commands from the **SNIPPETS.txt** file.

3. All lab steps are done on the NGINX lab system unless explicitly stated.

4. Determine the version of NGINX.

```
$ nginx -v
```

5. Test the NGINX configuration files before you make any changes.

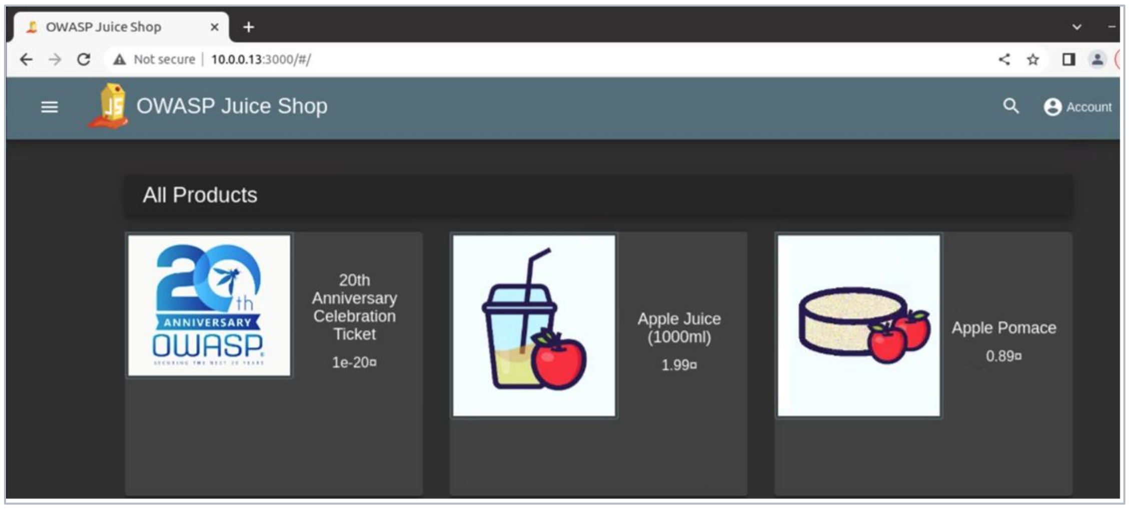
```
$ sudo nginx -t
```

6. View which NGINX processes are running. There should be one primary process and several worker processes.

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

7. Open a Chrome browser to test accessing the Juice Shop web application directly by entering the IP address for your **Server Container** lab system (10.0.0.13) and port 3000 which is where the Juice Shop application is running.

```
http://10.0.0.13:3000
```



8. Open another tab in the Chrome browser and then enter the IP address for your **NGINX** lab system (10.0.0.11). You are accessing the Juice Shop application via NGINX rather than going directlyto the Server Container lab system where the Juice Shop application is running.

```
http://10.0.0.11
```

9. Why do you not need to give port 3000 when accessing the Juice Shop application from the **NGINX** lab system?

10. To answer this, view the application configuration file **/etc/nginx/conf.d/juice.conf** looking at the **proxy_pass** and **upstream** context where the upstream server is defined as **10.0.0.13:3000** which is the **Server Container** lab system where the Juice Shop application is running.

    ```
    $ more /etc/nginx/conf.d/juice.conf
    ```

```
upstream juice_server {
   server 10.0.0.13:3000;
}

server {
    listen 80;
    root /usr/share/nginx/html;
    access_log /var/log/nginx/access.log combined;

    location / {
        proxy_pass http://juice_server;
    }

    error_page 500 502 503 504 /50x.html;

    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

```
upstream juice_server {
    server 10.0.0.13:3000;
}

server {
    listen 80;
    root /usr/share/nginx/html;
    access_log /var/log/nginx/access.log combined;

    location / {
        proxy_pass http://juice_server;
    }

    error_page 500 502 503 504 /50x.html;

    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

NOTE
From now on you will access the Juice Shop application using the NGINX lab system (10.0.0.11).

Expected Results

After logging into your NGINX lab system you will verify that NGINX Plus is already installed and that there is one NGINX master process along with one or more NGINX worker processes. You will test accessing the Juice Shop web application by going directly to the Server Container lab system using port 3000 and by using NGINX to access the Juice Shop application. You will view the contents of the /etc/nginx/conf.d/juice.conf file which has an upstream configured to access the Juice Shop application on the Server Container lab system.

Chapter 2

Labs - Setting up an HTTPS Server

Lab Contents

1. **Set DNS Variable**
2. **Generate Certs & Keys**
3. **Configure HTTPS Server**
4. **Configure SSL cipher, forward secrecy for Diffie-Hellman and test more secure NGINX configuration**

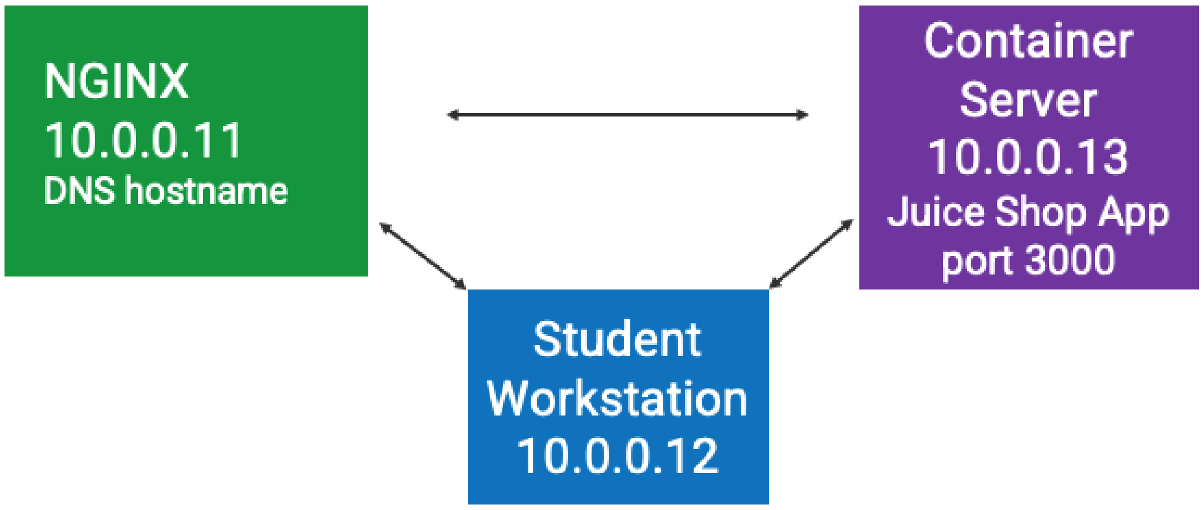
Estimated time to complete labs: **[XX] minutes**

Lab Credentials and IP Address List

LAB SYSTEM	IP ADDRESS
NGINX	10.0.0.11
Student Workstation	10.0.0.12
Container Server	10.0.0.13

Lab Network Topology

Lab Network Topology



There are 3 lab systems: NGINX, Student Workstation and Container Server. The NGINX lab system has NGINX Plus R28 installed. The Container Server system has the OWASP Juice Shop application running on port 3000 using Kubernetes. The Student Workstation lab system is an Ubuntu Linux system that is a graphical system used to test access to the NGINX lab system.

Lab 1: Generating Certs and Keys

Estimated time for completion: **XX minutes**

Requirements

Overview

In this exercise, you generate certificates for the NGINX system in order to configure HTTPS for accessing the web application Juice Shop.

Objectives

At the end of this lab you will be able to:

- Create a private CA Root Authority certificate
- Create a public and private certificate for the NGINX lab system
- Set permissions for cert files
- Add the NGINX directives for certificates

Steps

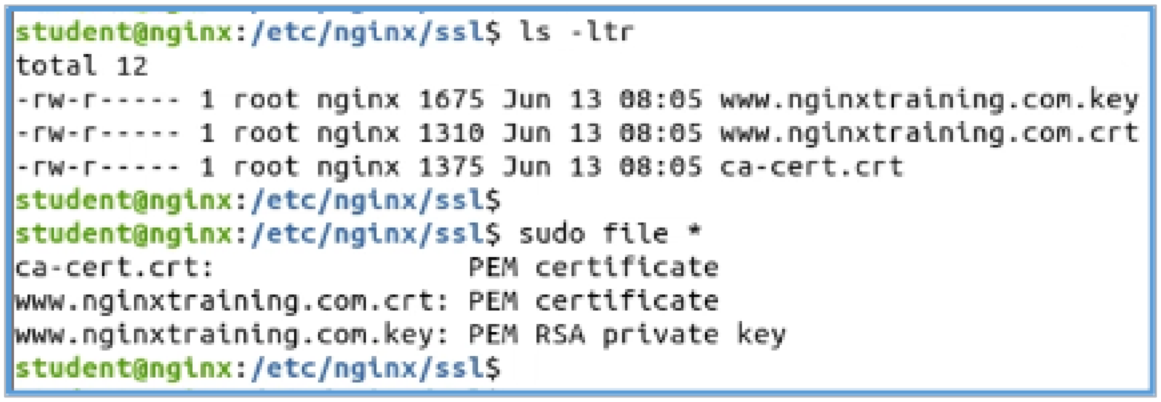
1. Create a private key for the CA Root Authority, generate the X509 certificate for the CA Root Authority, and create the public and private certs for your NGINX system. The ***create_certs*** script will create the CA Root Authority certs, and certs for a DNS name that represents your NGINX lab system **www.nginxtraining.com**.

```
$ cd ~/ssl
$ less create_certs.sh
$ ./create_certs.sh www.nginxtraining.com
$ ls -ltr
```

NOTE
Using self-signed certificates or a certificate signed by a private CA is for training or testing purposes. In a production environment you would have certs signed by a legitimate CA Root Authority.

2. List the permissions, owner and group of the cert files created. Change the group to nginx for the newly created certificate files and change the permissions as follows.

```
$ sudo cp *crt www*key /etc/nginx/ssl
$ cd /etc/nginx/ssl
$ sudo chgrp nginx *
$ sudo chmod 640 *
$ ls -ltr
$ sudo file *
```



```
student@nginx:/etc/nginx/ssl$ ls -ltr
total 12
-rw-r----- 1 root nginx 1675 Jun 13 08:05 www.nginxtraining.com.key
-rw-r----- 1 root nginx 1310 Jun 13 08:05 www.nginxtraining.com.crt
-rw-r----- 1 root nginx 1375 Jun 13 08:05 ca-cert.crt
student@nginx:/etc/nginx/ssl$
student@nginx:/etc/nginx/ssl$ sudo file *
ca-cert.crt:               PEM certificate
www.nginxtraining.com.crt: PEM certificate
www.nginxtraining.com.key: PEM RSA private key
student@nginx:/etc/nginx/ssl$
```

NOTE
The ca-cert.crt file is in PEM format. When using Dynamic certificate loading read access is required.

3. Before making changes, backup the original NGINX SSL configuration file /etc/nginx/ssl-configs/ssl-params.conf

```
$ sudo cp /etc/nginx/ssl-configs/ssl-params.{conf,orig}
```

4. Edit the ssl-params.conf file to add the **ssl_certificate** directives.

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

```
ssl_trusted_certificate /etc/nginx/ssl/ca-cert.crt;
ssl_certificate /etc/nginx/ssl/www.nginxtraining.com.crt;
ssl_certificate_key /etc/nginx/ssl/www.nginxtraining.com.key;
```

NOTE
The ssl_trusted_certificate directive needs to be before the ssl_certificate and ssl_certificate_key directives.

5. Save the file and exit the editor.

6. Test that the configuration syntax has no errors.

```
$ sudo nginx -t
```

Expected Results

You should now have the appropriate certificates set up for NGINX to access the Juice Shop application when HTTPS is configured next.

Lab 2: Configuring an HTTPS Server

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- You must have a valid certificate and private key for your NGINX lab system.

Overview

In this exercise, you configure HTTPS access to the Juice Shop app via NGINX

Objectives

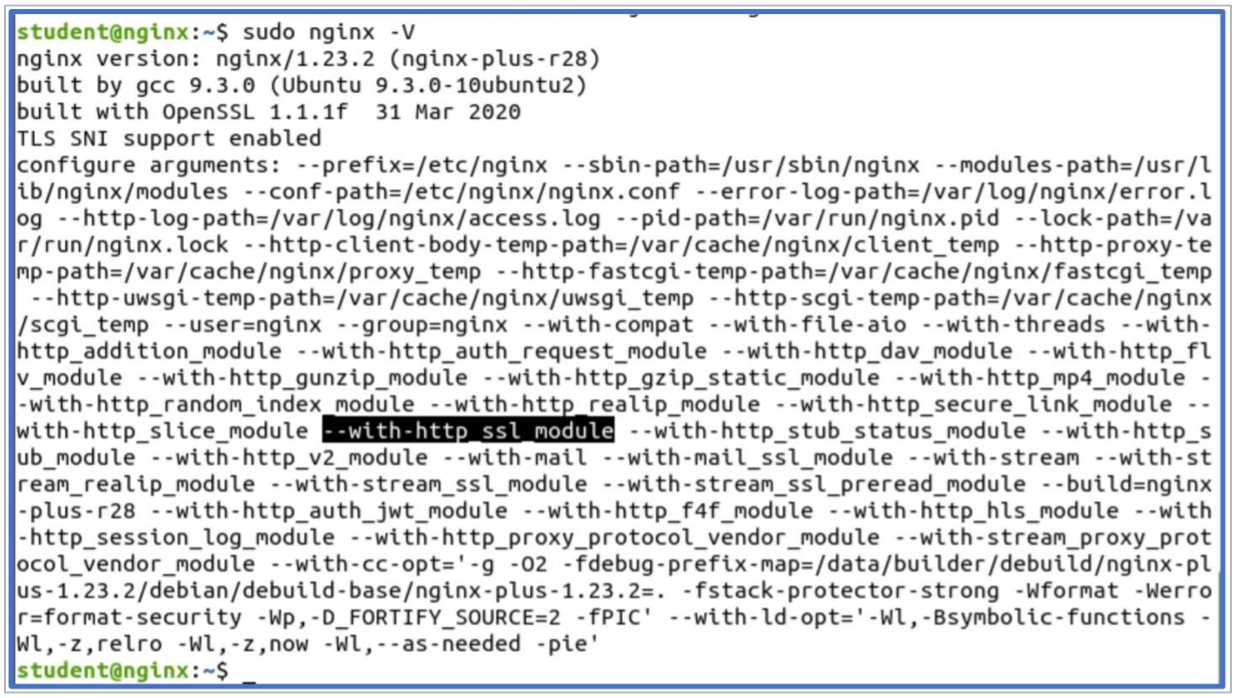
At the end of this lab you will be able to:

- Configure a server context listening on port 443 for HTTPS
- Set header values to be passed to the upstreams

Steps

1. Verify **NGINX** has support for HTTPS.

```
$ sudo nginx -V
```



```
student@nginx:~$ sudo nginx -V
nginx version: nginx/1.23.2 (nginx-plus-r28)
built by gcc 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
built with OpenSSL 1.1.1f  31 Mar 2020
TLS SNI support enabled
configure arguments: --prefix=/etc/nginx --sbin-path=/usr/sbin/nginx --modules-path=/usr/l
ib/nginx/modules --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.l
og --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --lock-path=/va
r/run/nginx.lock --http-client-body-temp-path=/var/cache/nginx/client_temp --http-proxy-te
mp-path=/var/cache/nginx/proxy_temp --http-fastcgi-temp-path=/var/cache/nginx/fastcgi_temp
 --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp --http-scgi-temp-path=/var/cache/nginx
/scgi_temp --user=nginx --group=nginx --with-compat --with-file-aio --with-threads --with-
http_addition_module --with-http_auth_request_module --with-http_dav_module --with-http_fl
v_module --with-http_gunzip_module --with-http_gzip_static_module --with-http_mp4_module -
-with-http_random_index_module --with-http_realip_module --with-http_secure_link_module --
with-http_slice_module --with-http_ssl_module --with-http_stub_status_module --with-http_s
ub_module --with-http_v2_module --with-mail --with-mail_ssl_module --with-stream --with-st
ream_realip_module --with-stream_ssl_module --with-stream_ssl_preread_module --build=nginx
-plus-r28 --with-http_auth_jwt_module --with-http_f4f_module --with-http_hls_module --with
-http_session_log_module --with-http_proxy_protocol_vendor_module --with-stream_proxy_prot
ocol_vendor_module --with-cc-opt='-g -O2 -fdebug-prefix-map=/data/builder/debuild/nginx-pl
us-1.23.2/debian/debuild-base/nginx-plus-1.23.2=. -fstack-protector-strong -Wformat -Werro
r=format-security -Wp,-D_FORTIFY_SOURCE=2 -fPIC' --with-ld-opt='-Wl,-Bsymbolic-functions -
Wl,-z,relro -Wl,-z,now -Wl,--as-needed -pie'
student@nginx:~$ _
```

2. Before making any changes, backup the juice.conf configuration file.

```
$ sudo cp /etc/nginx/conf.d/juice.{conf,orig}
```

3. Edit the Juice Shop configuration file.

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

4. After the **listen** directive, add a **return** directive that forces all http traffic to **https** and close this server context by adding a right brace **}**. Do not delete any other lines in the file as we are not finished editing.

```
server {
    listen 80;
    return 301 https://$host$request_uri;
}
  root /usr/share/nginx/html
```

5. After the close of the **return** directive, create a new server context listening on port **443**, with the **ssl** parameter enabled and make this the default server.

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

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

6. After the **listen** directive, add the **server_name** directive with the DNS name for your NGINX system, www.nginxtraining.com.

```
server {
   listen 443 ssl default_server;
  server_name www.nginxtraining.com;
  root /usr/share/nginx/html;
  access_log /var/log/nginx/access.log combined;
...
```

7. After the **root** directive, add an **include** directive for accessing your custom security settings file /etc/nginx/ssl-configs/ssl-params.conf.

```
server {
  listen 443 ssl default_server;
  server_name www.nginxtraining.com;
  root /usr/share/nginx/html;
  include /etc/nginx/ssl-configs/ssl-params.conf;
  access_log /var/log/nginx/access.log combined;
...
```

8. After the **access_log** directive, add the following **proxy_set_header** directives.

```
server {
  listen 443 ssl default_server;
  server_name www.nginxtraining.com;
  root /usr/share/nginx/html;
  include /etc/nginx/ssl-configs/ssl-params.conf;
  access_log /var/log/nginx/access.log combined;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header Host $host;
```

9. Save the and exit the juice.conf configuration file.

10. Test your NGINX updated configuration files for any syntax errors.

```
$ sudo nginx -t
```

Expected Results

You will now have a server context set up for HTTPS.

Lab 3: Set SSL Ciphers

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- You must have a valid certificate and private key for your NGINX lab system.
- You must have a server block defined for Juice Shop to handle HTTPS requests.

Overview

In this lab, edit the NGINX configuration file for the Juice Shop application to configure specific SSL ciphers.

Objectives

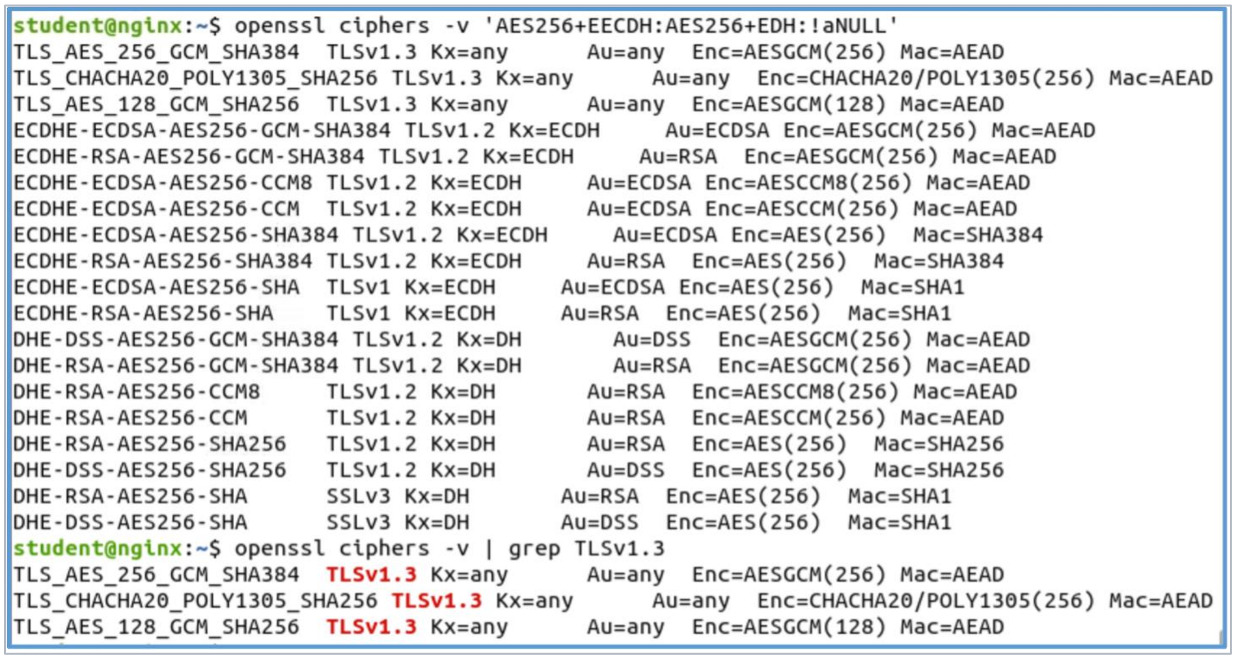
At the end of this lab you will be able to:

- Specify which SSL cipher algorithms for NGINX to use.

Steps

1. Use openssl to list the ciphers that match our request with a complete description of the protocol version, key exchange, authentication, encryption and mac algorithms used along with any key size restrictions.

```
$ openssl ciphers -v 'AES256+EECDH:AES256+EDH:!aNULL'
$ openssl ciphers -v | grep TLSv1.3
```



```
student@nginx:~$ openssl ciphers -v 'AES256+EECDH:AES256+EDH:!aNULL'
TLS_AES_256_GCM_SHA384  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any      Au=any  Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-CCM8 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESCCM8(256) Mac=AEAD
ECDHE-ECDSA-AES256-CCM  TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AESCCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA384
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA384
ECDHE-ECDSA-AES256-SHA  TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA1
ECDHE-RSA-AES256-SHA    TLSv1 Kx=ECDH     Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=DSS  Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH       Au=RSA  Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-CCM8     TLSv1.2 Kx=DH       Au=RSA  Enc=AESCCM8(256) Mac=AEAD
DHE-RSA-AES256-CCM      TLSv1.2 Kx=DH       Au=RSA  Enc=AESCCM(256) Mac=AEAD
DHE-RSA-AES256-SHA256   TLSv1.2 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA256
DHE-DSS-AES256-SHA256   TLSv1.2 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA256
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-SHA      SSLv3 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA1
student@nginx:~$ openssl ciphers -v | grep TLSv1.3
TLS_AES_256_GCM_SHA384  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any      Au=any  Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256  TLSv1.3 Kx=any      Au=any  Enc=AESGCM(128) Mac=AEAD
```

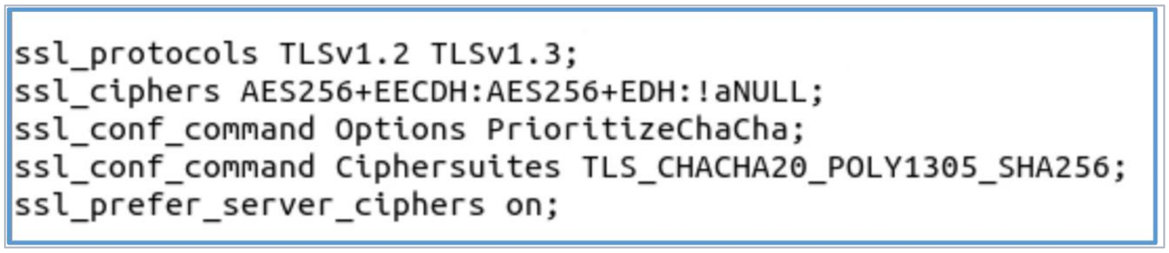
NOTE
Remember to copy the command from the SNIPPETS.txt file or type in the quotes manually.

An "!" means that the cipher is deleted from the list. In this lab we are removing !aNULL which is the anonymous Diffie-Hellman algorithm which is vulnerable to MITM (Man In The Middle) attacks.

2. Edit the configuration file **ssl-params.conf** to add your allowed/preferred ciphers for both TLSv1.2 and TLSv1.3.

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

ssl_ciphers AES256+EECDH:AES256+EDH:!aNULL;
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;



```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers AES256+EECDH:AES256+EDH:!aNULL;
ssl_conf_command Options PrioritizeChaCha;
ssl_conf_command Ciphersuites TLS_CHACHA20_POLY1305_SHA256;
ssl_prefer_server_ciphers on;
```

3. Save the and exit the juice.conf configuration file.

4. Test your NGINX updated configuration files for any syntax errors.

   ```
   $ sudo nginx -t
   ```

Expected Results

You should have 2 server blocks in your NGINX juice.conf file, one that listens on port 80 for HTTP traffic and one that listens on port 443 for HTTPS traffic. All HTTP traffic will be redirected to the server block for HTTPS.

Lab 4: Configure Forward Secrecy

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- You must have a valid certificate and private key for your NGINX lab system.
- You must have a server block defined for Juice Shop to handle HTTPS requests.

Objectives

At the end of this lab you will be able to:

- Configure Forward Secrecy in NGINX for Diffie-Hellman

Steps

1. Generate a new Diffie-Hellman Key-Exchange. Use the -dsaparam flag in order to avoid testing for primality in all 4 million numbers.

   ```
   $ sudo openssl dhparam -dsaparam -out /etc/nginx/dhparam.pem 4096
   ```

2. Edit the configuration file ssl-params.conf to add the ssl_dhparam directive with the dhparam.pem file you created in the previous step.

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

   ```
   ssl_dhparam /etc/nginx/dhparam.pem ;
   ```

3. Save the and exit the file.

4. Reload NGINX to use your updated configuration files.

   ```
   $ sudo nginx -t && sudo nginx -s reload
   ```

5. Edit the file /etc/hosts to add the DNS name www.nginxtraining.com to be associated with your NGINX lab system 10.0.0.11.

   ```
   $ sudo vim /etc/hosts
   ```

   ```
   10.0.0.11     www.nginxtraining.com
   ```

NOTE
Typically, if you are testing you would not edit system files but would instead use command options to handle your test case. For the curl command you can add the option **--resolve www.nginxtraining.com:443:10.0.0.11** but there may be limited information returned so for our training purposed adding the NGINX DNS name and IP address to /etc/hosts is preferable.

Expected Results

Your NGINX configuration now uses a specific SSL cipher algorithm and is set up to use Forward Secrecy with Diffie-Hellman.

Lab 5: Testing HTTPS Configuration

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- You must have a valid certificate and private key for your NGINX lab system.
- You must have a server block defined for Juice Shop to handle HTTPS requests.

Objectives

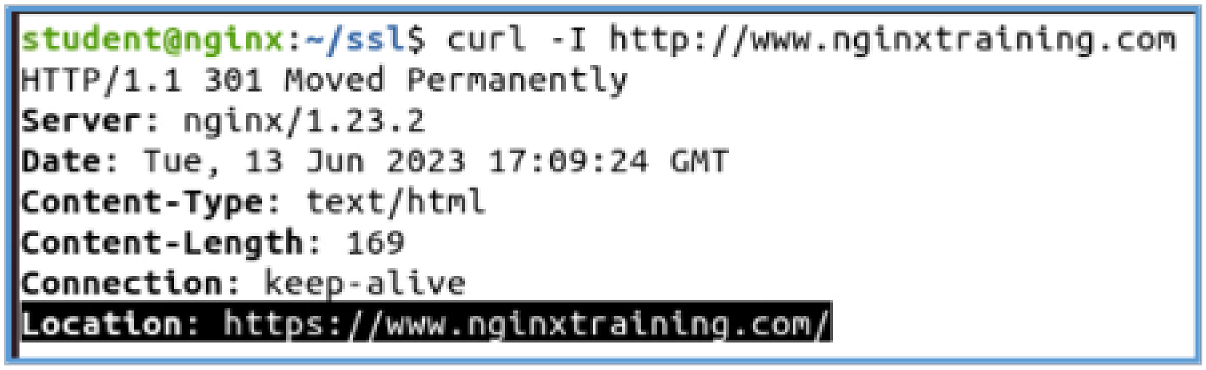
At the end of this lab you will be able to:

- Test HTTPS configuration

Steps

1. Perform a curl test using HTTP. The output should have a 301 Moved Permanently message.

```
$ curl -I http://www.nginxtraining.com
```

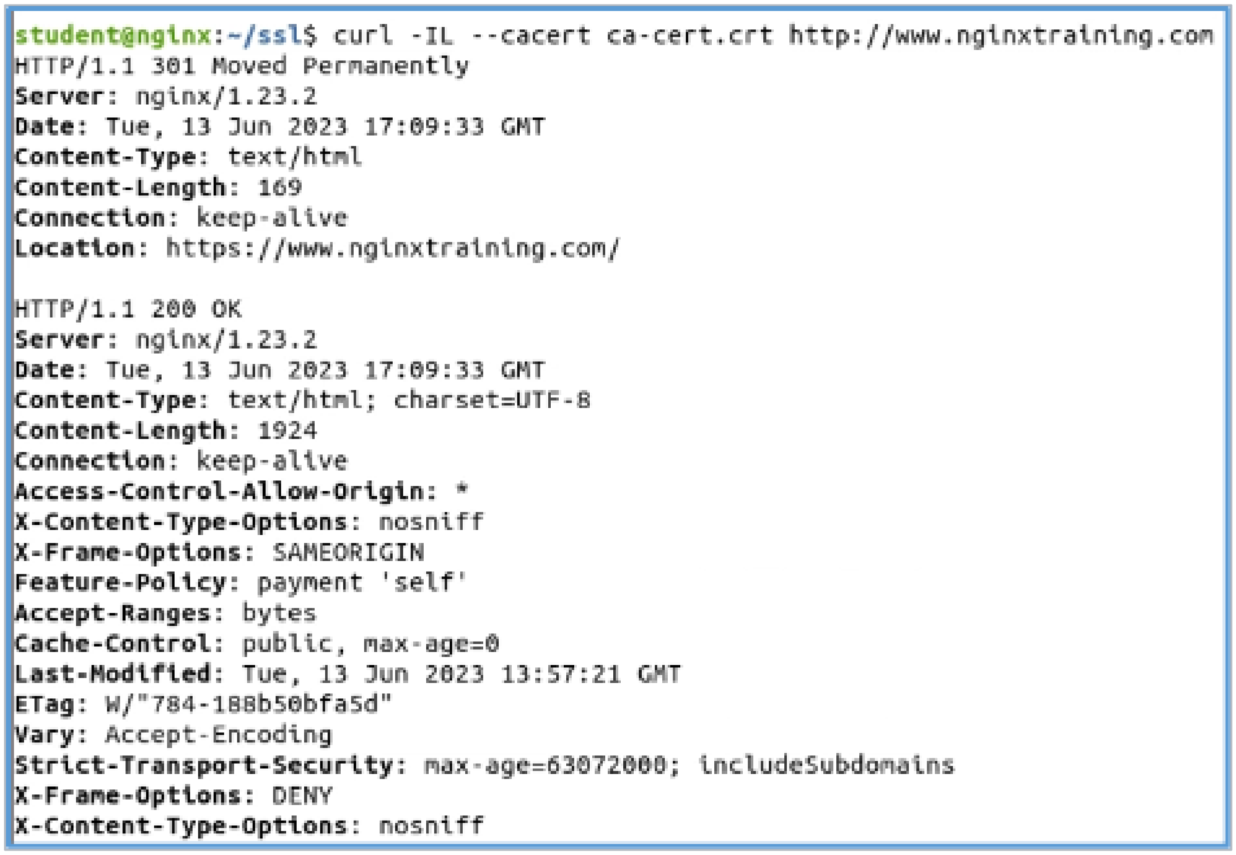


```
student@nginx:~/ssl$ curl -I http://www.nginxtraining.com
HTTP/1.1 301 Moved Permanently
Server: nginx/1.23.2
Date: Tue, 13 Jun 2023 17:09:24 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://www.nginxtraining.com/
```

NOTE
This works for curl so you don't have to use the -k option, but in the browser it still gives ~~https~~ which means it's not secure/trusted. Depending on which browser you are using you can add your certificate to the browser trust but since in production you will not be using self-signed certs you can ignore these warnings in our training lab environment.

2. Perform another curl test using HTTP with the -L option to follow to the new location where the requested page has moved. These are two ways to not have curl complain about not being able to verify the certificate from using either a self-signed cert or like in our lab using a cert that was created from a private CA. The results of both commands are the same.

```
$ cd ~/ssl
$ curl -IL --cacert ca-cert.crt http://www.nginxtraining.com
$ curl -ILk http://www.nginxtraining.com
```



```
student@nginx:~/ssl$ curl -IL --cacert ca-cert.crt http://www.nginxtraining.com
HTTP/1.1 301 Moved Permanently
Server: nginx/1.23.2
Date: Tue, 13 Jun 2023 17:09:33 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://www.nginxtraining.com/

HTTP/1.1 200 OK
Server: nginx/1.23.2
Date: Tue, 13 Jun 2023 17:09:33 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1924
Connection: keep-alive
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 13 Jun 2023 13:57:21 GMT
ETag: W/"784-188b50bfa5d"
Vary: Accept-Encoding
Strict-Transport-Security: max-age=63072000; includeSubdomains
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
```

NOTE
Your output should have a 301 Moved Permanently but will also follow to the new location (200 OK) and that requires the certificate to be verified which is why we used the option --cacert to specify the CA Root Authority cert that our cert & key files were referenced with or the -k option to skip verification. Another way to avoid the verification issue is to put the CA cert you created into the system certificates store; for an Ubuntu system that would include copying the file ca-cert.crt to /usr/local/share/ca-certificates and then running "sudo update-ca-certificates".

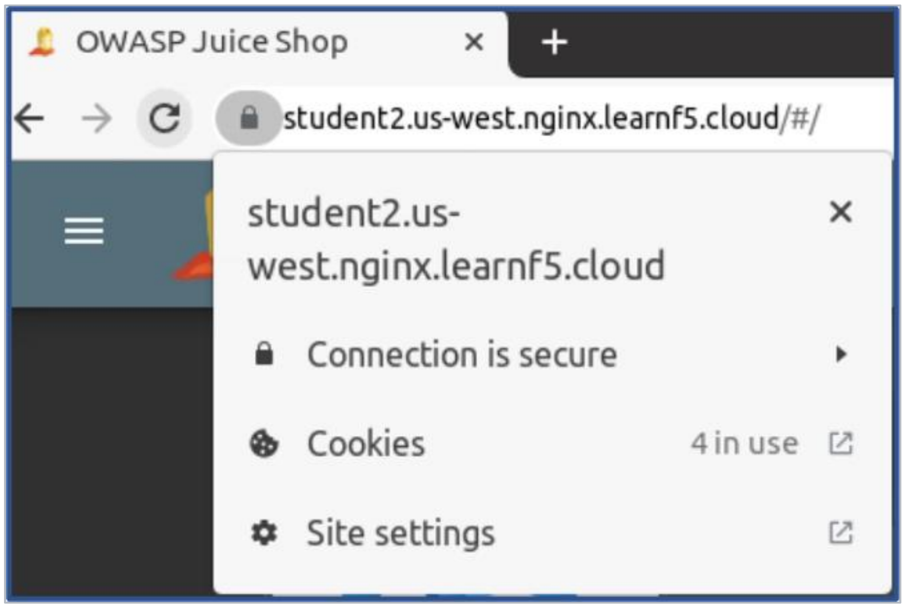
3. Perform a curl test using **HTTPS**. Your output should have a **200 OK** response.

```
$ curl -I https://www.nginxtraining.com
```

TROUBLESHOOTING
If you receive a 301 Moved Permanently response then make sure that the curl command is using **https** not http.

4. In your browser use **http://www.nginxtraining.com** to verify if the webpage now uses **https**. Because our certificate is signed by a private CA the browser can't verify which means the ~~https~~ will be crossed out and you need to click on the Advanced box at the bottom left and then click on the Proceed to www.nginxtraining.com (unsafe) link, which will take you to the Juice Shop web app via your NGINX lab system.

NOTE
If you were using a public (legitimate) CA certificate, then you would get a lock symbol or https.


OWASP Juice Shop
student2.us-west.nginx.learnf5.cloud/#/
student2.us-west.nginx.learnf5.cloud
Connection is secure
Cookies
4 in use
Site settings

Expected Results

Your NGINX configuration now uses HTTPS with a specific SSL cipher algorithm and is set up to use Forward Secrecy with Diffie-Hellman.

Labs - Secure Upstream Traffic

Lab Contents

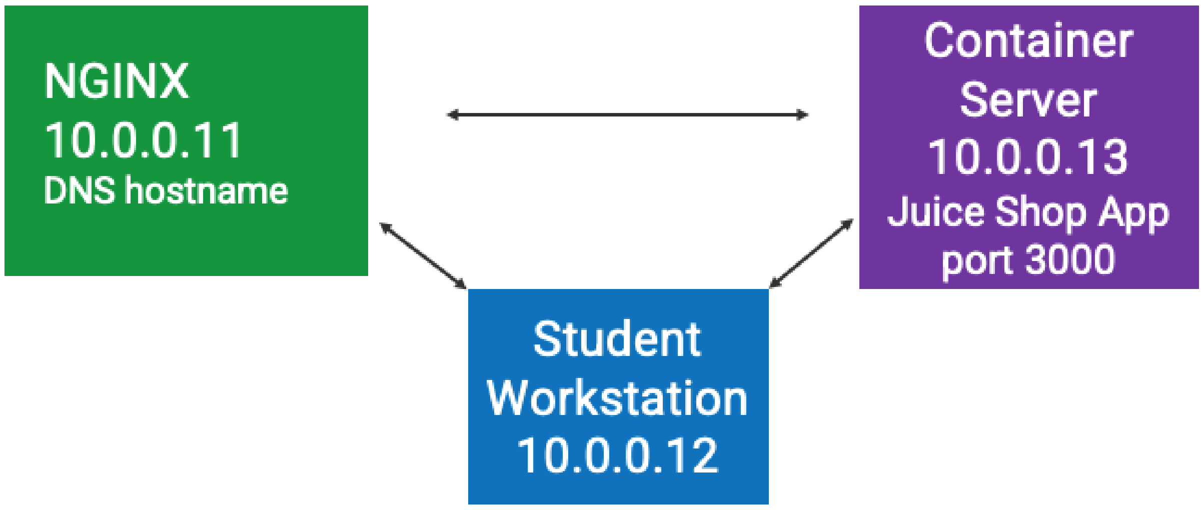
1. **Secure Upstream Traffic**
2. **Configure the NGINX Plus API Dashboard**
3. **Configure NGINX Access and Error Logging**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology- Possibly Delete Topic?

Lab Network Topology

Lab Network Topology



There are 3 lab systems: NGINX, Student Workstation and Container Server. The NGINX lab system has NGINX Plus R28 installed. The Container Server system has the OWASP Juice Shop application running on port 3000 using Kubernetes. The Student Workstation lab system is an Ubuntu Linux system that is a graphical system used to test access to the NGINX lab system.

Lab 1: Secure Upstream Traffic

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- You must have a valid certificate and private key for your NGINX lab system.
- You must have a server block defined for Juice Shop to handle HTTPS requests.

Objectives

At the end of this lab you will be able to:

- Secure upstream traffic with ssl parameters
- Configure the NGINX API Dashboard to gather and view metrics
- Configure NGINX access and error logs

Steps

1. Backup the configuration file **api_server.conf**

   $ sudo cp /etc/nginx/conf.d/api_server.{conf,orig}

2. Edit the configuration file **api_server.conf**

   $ sudo vim /etc/nginx/conf.d/api_server.conf

3. Enable HTTPS on port **8080** using the parameter **ssl.**

```
server {
    listen 8080 ssl;
    root /usr/share/nginx/html;
    ...
}
```

4. In the server block with ssl, create a **location** with the following proxy settings.

```
location / {
proxy_pass https://api_server;
proxy_ssl_trusted_certificate /etc/nginx/ssl/ca-cert.crt;
  proxy_ssl_verify off;
}
```

5. Save the api_server.conf file and then reload NGINX to use your updated configuration files.

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

6. Test your configuration with curl. Your output should get a **502 Bad Gateway** response because you are trying to access upstreams that are not configured.

   ```
   $ curl -k https://www.nginxtraining.com:8080/
   ```

7. View the contents of the **proxy-ssl-params.conf** file which defines the SSL protocols and HTTP version.

   ```
   $ cat /etc/nginx/ssl-configs/proxy-ssl-params.conf
   ```

8. Edit the **api_server.conf** file to add the **proxy-ssl-params.conf** file in the location / context.

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

   location / {
      include /etc/nginx/ssl-configs/proxy-ssl-params.conf;
     ...
   }
   ```

9. Continue editing the api_server.conf file and for both server contexts, enable **ssl** and include the file **ssl-params.conf**

> NOTE
> This is **not the same pathname** as the previous step. There is no **proxy** in the name so be sure to edit the filename if you copy and paste the previous command or just copy from the Snippets file!

```
listen 8081 ssl;
include /etc/nginx/ssl-configs/ssl-params.conf;
…
```

```
listen 8082 ssl;
include /etc/nginx/ssl-configs/ssl-params.conf;
```

10. Save the configuration file and reload NGINX.

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

11. Test the upstream backend systems using the curl command.

    ```
    $ cd ~/ssl
    $ curl --cacert ca-cert.crt https://www.nginxtraining.com:8080
    $ curl --cacert ca-cert.crt https://www.nginxtraining.com:8080
    $ curl --cacert ca-cert.crt https://www.nginxtraining.com:8080
    ```

NOTE
Your output should be like "**this is app1**", then "**this is app2**" and then "**this is app1**" again. It does not matter which upstream server responds first since the default load balancing method is round robin so the backend servers should alternate serving the response. You might need to do more than three curl statements to see the Load Balancing start.

TROUBLESHOOTING:
If you are still getting a 502 Bad Gateway response message then make sure that you added "ssl" to both listen directives for ports 8081 and 8082 in the /etc/nginx/conf.d/api_server.conf configuration file and that you reload to use the updated configuration file.

Expected Results

The default round robin load balancing is set up for your application when using port 8080.

Lab 2: Setup the NGINX Plus API Dashboard

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- NGINX load balancing using the default method, round robin, is configured when using port 8080.

Overview

In this lab you will configure the NGINX Plus API Dashboard.

Objectives

At the end of this lab you will be able to:

- Access the NGINX Plus API Dashboard.

Steps

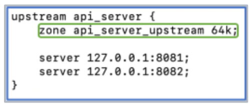
1. Edit the file /etc/nginx/conf.d/**api_server.conf**. In the context for 8080, create two new prefix locations: **api** and **dashboard**.

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

location /api {
    api;
    access_log off;
}

location /dashboard {
    root /usr/share/nginx/html;
    try_files $uri $uri.html /dashboard.html;
}
```

NOTE
For our lab purposes we are turning off the access log since it generates a fair amount of information whereas in production you may not want to turn off.

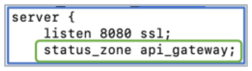
2. Continue editing the **api_server.conf** file, to gather status metrics add a **zone** directive in the upstream which is for the backends.

```
zone api_server_upstream 64k;
```



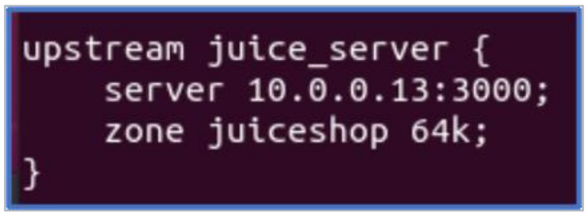
```
upstream api_server {
    zone api_server_upstream 64k;

    server 127.0.0.1:8081;
    server 127.0.0.1:8082;
}
```

3. Continue editing the **api_server.conf** file, to gather status metrics by adding a **status_zone** for the server listening on port 8080, which is for the front end, virtual servers. Save the file.



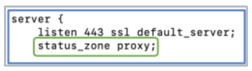
```
server {
    listen 8080 ssl;
    status_zone api_gateway;
```

4. To gather more status metrics, edit the file /etc/nginx/conf.d/**juice.conf**. Add a **zone** directive in the upstream juice_server.



```
upstream juice_server {
    server 10.0.0.13:3000;
    zone juiceshop 64k;
}
```

5. Continue editing the **juice.conf** file by adding a **status_zone** for the server listening on port 443.



```
server {
    listen 443 ssl default_server;
    status_zone proxy;
```

6. Save the file.

7. Test the syntax of your configuration files and when there are no errors reload the NGINX configuration.

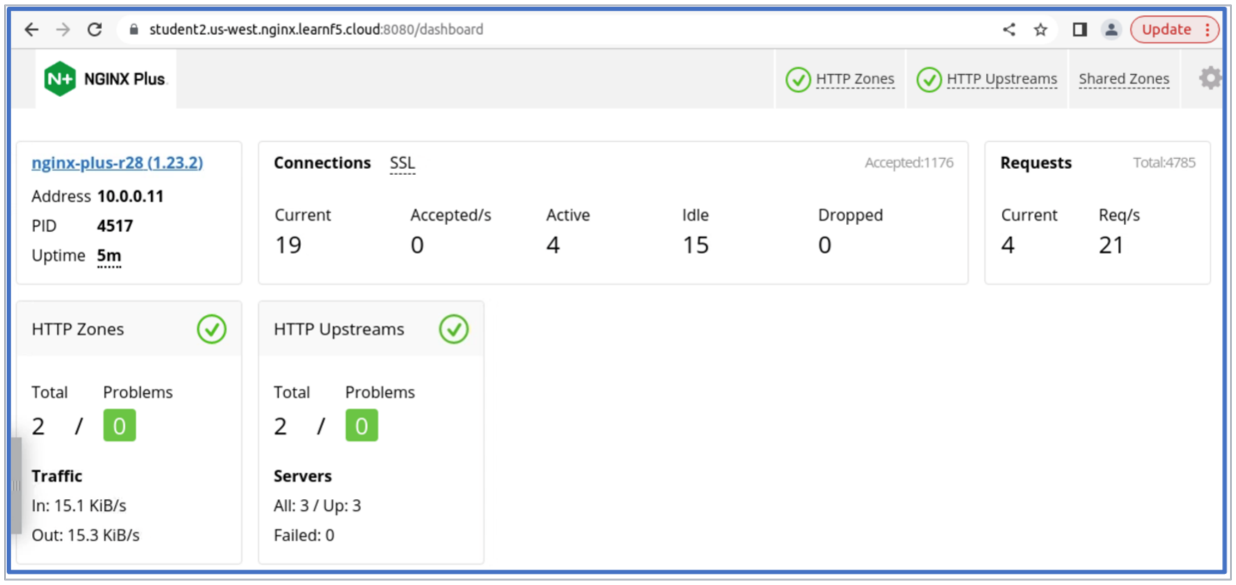
```
$ sudo nginx -t && sudo nginx -s reload
```

8. Send a few curl commands so that you have some newer data in your NGINX dashboard.

```
$ cd ~/ssl
$ curl --cacert ca-cert.crt https://www.nginxtraining.com:8080
$ curl --cacert ca-cert.crt https://www.nginxtraining.com:8080
```

9. Enter https://www.nginxtraining.com in a browser. Note that you will still get a note from the browser about not trusted and this is okay since we have not added our private Root CA to the browsers trust.

10. Access the NGINX dashboard from a browser using **https://www.nginxtraining.com:8080/dashboard**



nginx-plus-r28 (1.23.2)
Address 10.0.0.11
PID 4517
Uptime 5m

Connections SSL Accepted:1176

Current	Accepted/s	Active	Idle	Dropped
19	0	4	15	0

Requests Total:4785

Current	Req/s
4	21

HTTP Zones
Total 2 / Problems 0
Traffic
In: 15.1 KiB/s
Out: 15.3 KiB/s

HTTP Upstreams
Total 2 / Problems 0
Servers
All: 3 / Up: 3
Failed: 0

NOTE
If the HTTP Upstreams is yellow, it should eventually turn to green. You could send more requests by doing several curl -k https://www.nginxtraining.com:8080

11. In the NGINX dashboard click on the **HTTP Upstreams** tab to see both upstreams api_server and juice_server.

12. In the dashboard click on the **HTTP Zones** tab to see the zones you added to the api_server.conf and juice.conf files (api_gateway and proxy).

13. In your dashboard click on the **Shared Zones** tab to see the api_server_upstream zone you added to the api_server.conf file and the juiceshop zone in the juice.conf file.

14. To get back to the original dashboard view, click the **NGINX Plus logo** (top left of the dashboard).

Expected Results

The NGINX Plus Dashboard should be accessible to view upstreams, HTTP zones and shared zones.

Lab 3: Configure NGINX Logging

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- You will need cert files and a server context for HTTPS
- You will need the /etc/nginx/conf.d/juice.conf file.

Objectives

At the end of this lab you will be able to:

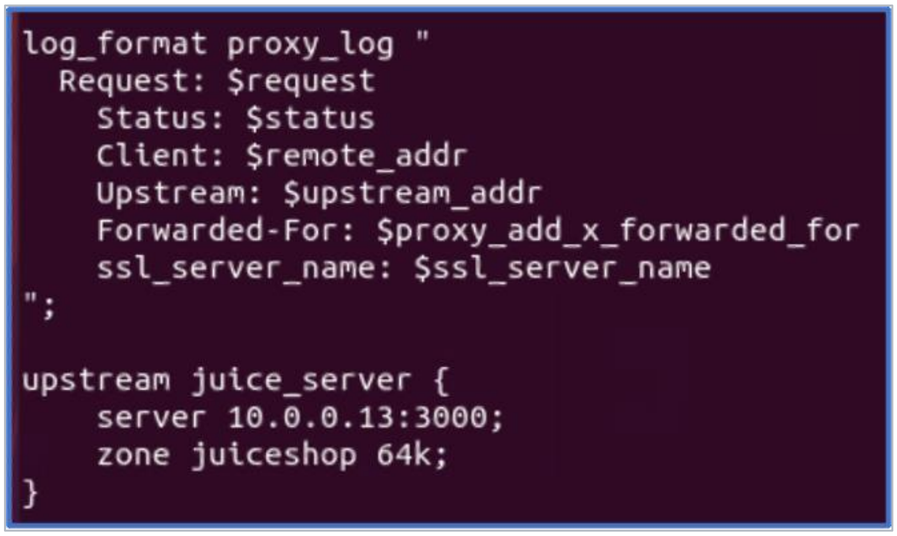
- Customize the format of the NGINX access log file.
- Access the NGINX access and error log files.

Steps

1. Edit the configuration file juice.conf.

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

2. Add a custom log format in the http context (at the very top of the file, outside of any server context). Make sure to copy from the SNIPPETS.txt file.



```
log_format proxy_log "
  Request: $request
    Status: $status
    Client: $remote_addr
    Upstream: $upstream_addr
    Forwarded-For: $proxy_add_x_forwarded_for
    ssl_server_name: $ssl_server_name
";

upstream juice_server {
    server 10.0.0.13:3000;
    zone juiceshop 64k;
}
```

3. Add the access and error log directives in the ssl server context and delete the original access_log directive using the combined format. Then save the file.

```
access_log /var/log/nginx/juice.access.log proxy_log;
error_log /var/log/nginx/juice.error.log info;
```

4. Test the syntax of your configuration files and when there are no errors reload the NGINX configuration.

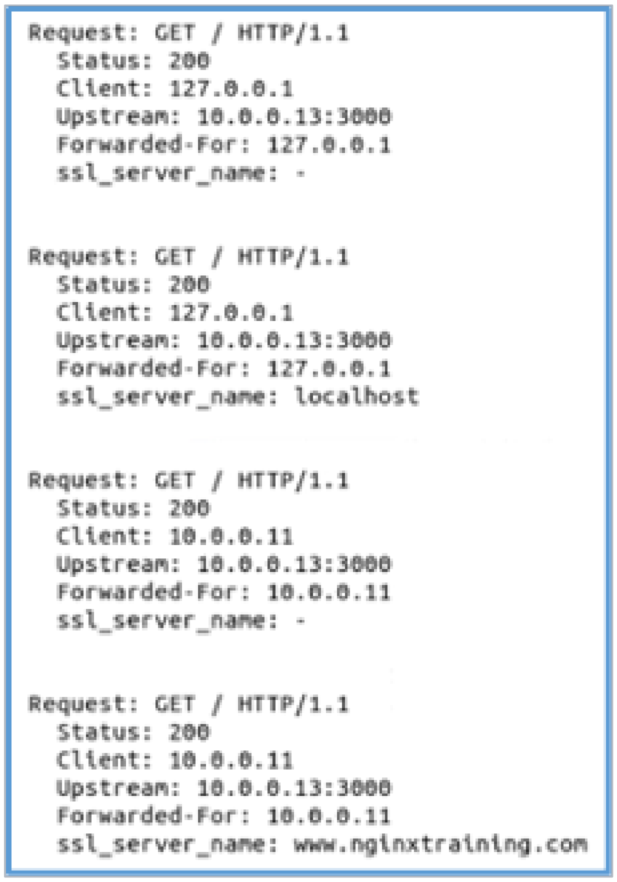
```
$ sudo nginx -t && sudo nginx -s reload
```

5. Open another terminal to view the NGINX access log file.

```
$ sudo tail -f /var/log/nginx/juice.access.log
```

6. Test the new log format using curl commands and focus on the values output in the variable *ssl_server_name* which comes from the SNI TLS handshake. View the **curl_script.sh** file to view the four curl commands and then run this script.

```
$ cat ~/curl_script.sh
$ ~/curl_script.sh
```



```
Request: GET / HTTP/1.1
  Status: 200
  Client: 127.0.0.1
  Upstream: 10.0.0.13:3000
  Forwarded-For: 127.0.0.1
  ssl_server_name: -


Request: GET / HTTP/1.1
  Status: 200
  Client: 127.0.0.1
  Upstream: 10.0.0.13:3000
  Forwarded-For: 127.0.0.1
  ssl_server_name: localhost


Request: GET / HTTP/1.1
  Status: 200
  Client: 10.0.0.11
  Upstream: 10.0.0.13:3000
  Forwarded-For: 10.0.0.11
  ssl_server_name: -


Request: GET / HTTP/1.1
  Status: 200
  Client: 10.0.0.11
  Upstream: 10.0.0.13:3000
  Forwarded-For: 10.0.0.11
  ssl_server_name: www.nginxtraining.com
```

NOTE
When using these values the variable ssl_server_name is only set when passing localhost and www.nginxtraining.com.

7. Use Control-C to exit the tail command.

Expected Results

You will now have a custom log format for the NGINX access log.

Chapter 3

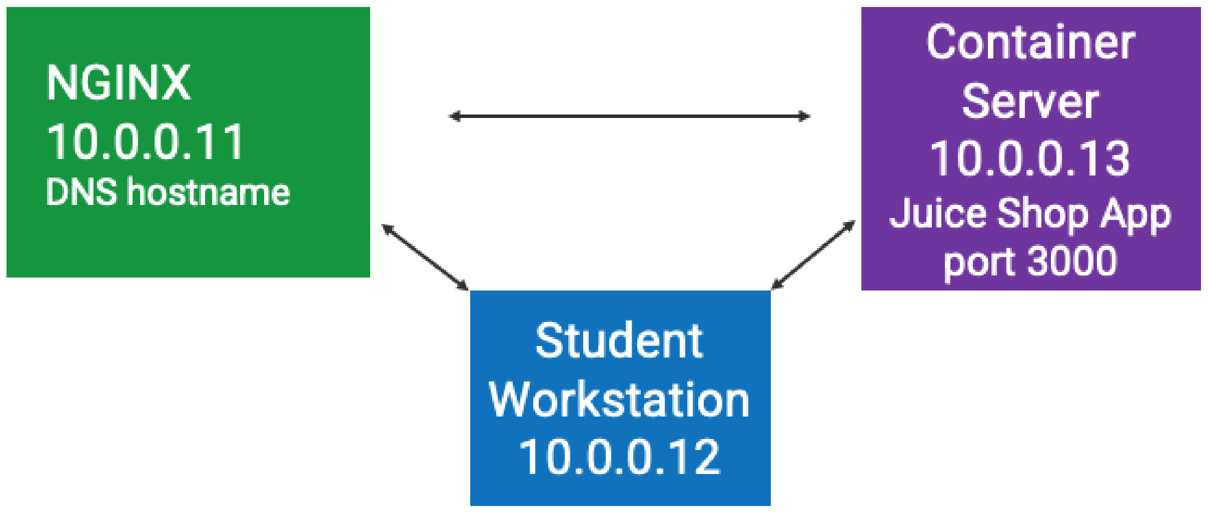
Labs - Set Limit Rates

Lab Contents

1. **Configure geo limits**
2. **Configure Limit Rates**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology



Lab 1: Configure NGINX limit rates

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- NGINX lab system with access to Juice Shop application running on another Linux lab system

Objectives

At the end of this lab you will be able to:

- Use the NGINX geo and map directives.
- Configure limit request rates in NGINX.

Steps

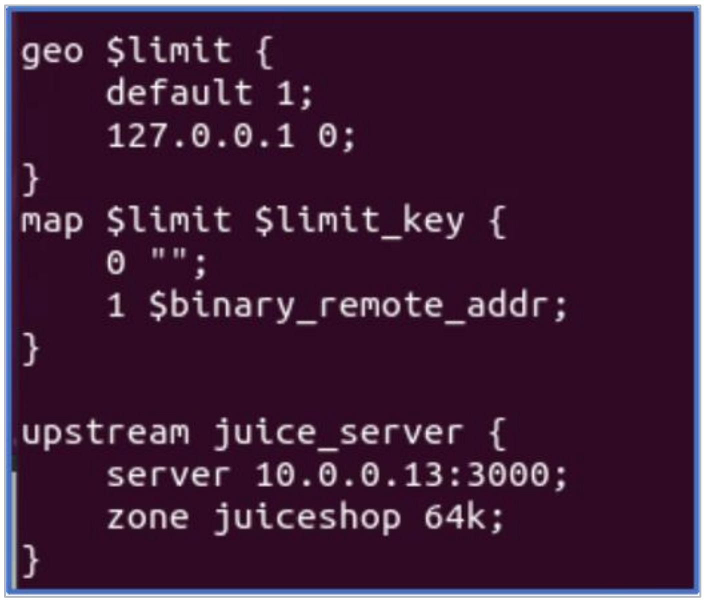
1. Create a copy of the juice.conf file before configuring limits in NGINX.

   ```
   $ sudo cp /etc/nginx/conf.d/juice.{conf,PRElimits}
   ```

2. Edit the **juice.conf** file to add a **geo** and **map** directive in the http context (for example at the top of the file outside of any server context). The **map** directive sets the **$limit_key** variable to the client's IP address (in byte form via the variable $binary_remote_addr).

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

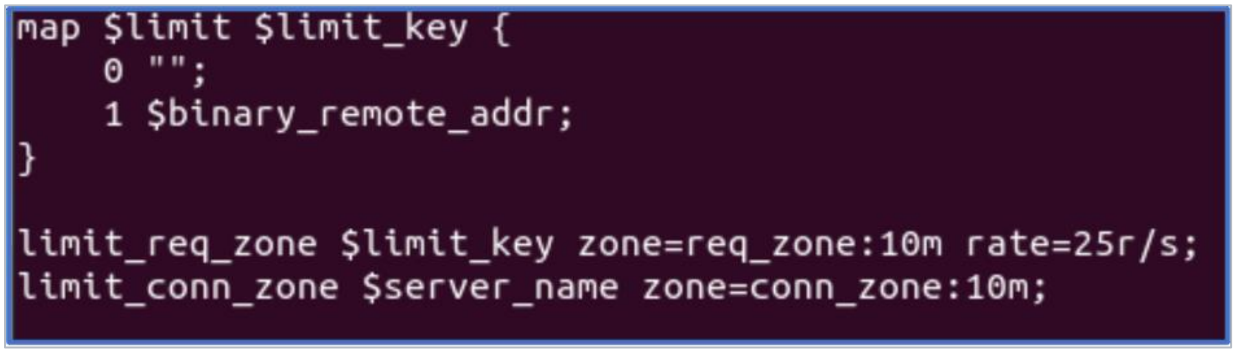
   ```
   geo $limit {
       default 1;
       127.0.0.1 0;
   }
   map $limit $limit_key {
       0 “”;
       1 $binary_remote_addr;
   }
   ```



```
geo $limit {
    default 1;
    127.0.0.1 0;
}
map $limit $limit_key {
    0 "";
    1 $binary_remote_addr;
}

upstream juice_server {
    server 10.0.0.13:3000;
    zone juiceshop 64k;
}
```

3. In the http context, after the map, create a **limit_req_zone** using the **$limit_key** variable and set the request rate to 25 requests per second and then create a **limit_conn_zone** using the variable $server_name using 10 megabytes for the number of connections.

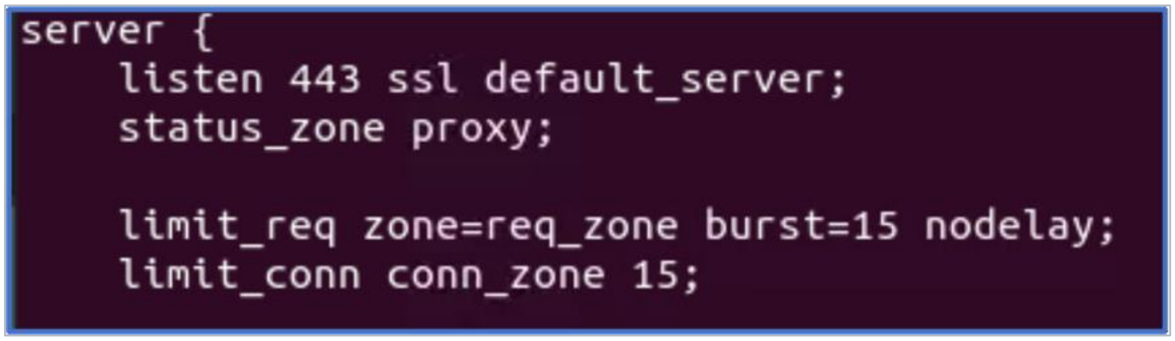
```
limit_req_zone $limit_key zone=req_zone:10m rate=25r/s;
limit_conn_zone $server_name zone=conn_zone:10m;
```



```
map $limit $limit_key {
    0 "";
    1 $binary_remote_addr;
}

limit_req_zone $limit_key zone=req_zone:10m rate=25r/s;
limit_conn_zone $server_name zone=conn_zone:10m;
```

4. In the ssl server context, set **limit_req** with a burst of 15, nodelay and use **req_zone** as the zone name. Also set the shared memory zone and the maximum allowed number of connections to 15 with the directive **limit_conn**.

```
limit_req zone=req_zone burst=15 nodelay;
limit_conn conn_zone 15;
```

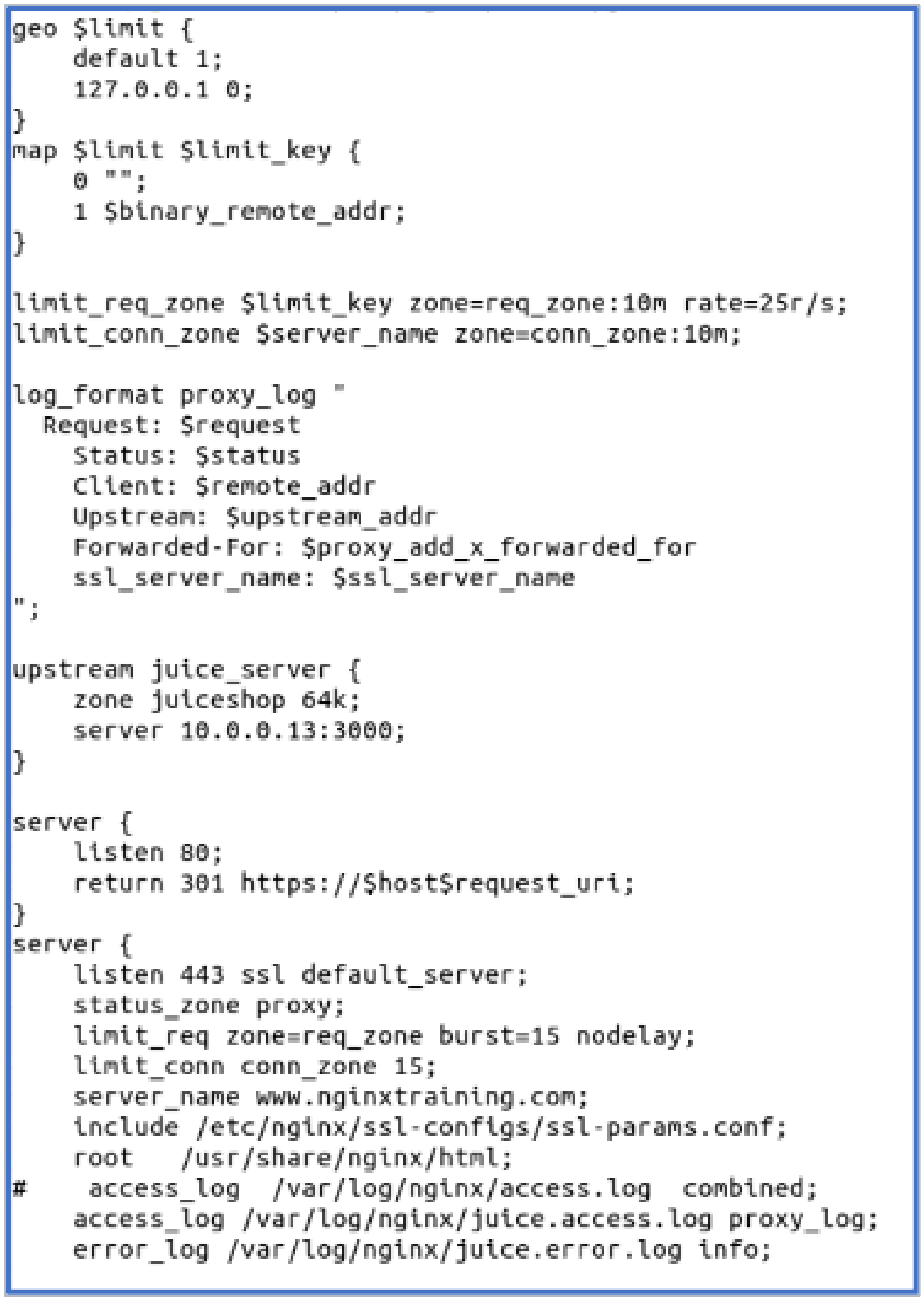


```
server {
    listen 443 ssl default_server;
    status_zone proxy;

    limit_req zone=req_zone burst=15 nodelay;
    limit_conn conn_zone 15;
```

5. Save the juice.conf file and then reload your NGINX configuration to make it active.

```
$ sudo nginx -t && sudo nginx -s reload
```

NOTE
All of the changes you made to the juice.conf file are shown below (but this is not the whole file):



```
geo $limit {
    default 1;
    127.0.0.1 0;
}
map $limit $limit_key {
    0 "";
    1 $binary_remote_addr;
}

limit_req_zone $limit_key zone=req_zone:10m rate=25r/s;
limit_conn_zone $server_name zone=conn_zone:10m;

log_format proxy_log "
  Request: $request
    Status: $status
    Client: $remote_addr
    Upstream: $upstream_addr
    Forwarded-For: $proxy_add_x_forwarded_for
    ssl_server_name: $ssl_server_name
";

upstream juice_server {
    zone juiceshop 64k;
    server 10.0.0.13:3000;
}

server {
    listen 80;
    return 301 https://$host$request_uri;
}
server {
    listen 443 ssl default_server;
    status_zone proxy;
    limit_req zone=req_zone burst=15 nodelay;
    limit_conn conn_zone 15;
    server_name www.nginxtraining.com;
    include /etc/nginx/ssl-configs/ssl-params.conf;
    root   /usr/share/nginx/html;
#    access_log  /var/log/nginx/access.log  combined;
    access_log /var/log/nginx/juice.access.log proxy_log;
    error_log /var/log/nginx/juice.error.log info;
```

NOTE
We will test these limits in the next lab exercise.

Expected Results

Your NGINX configuration has limit rates set for maximum requests.

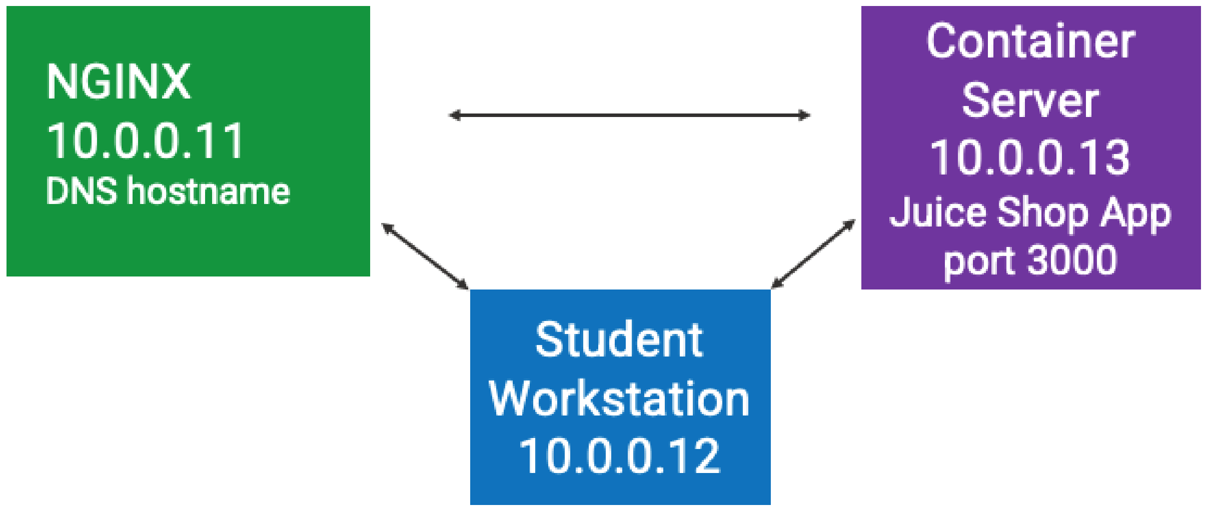
Labs - Logs

Lab Contents

1. **Access Log**
2. **Logging Limit Rates**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology



Lab 1: NGINX Access and Error Log Files

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- Have limit request rates set in the juice.conf configuration file on the NGINX lab system.

Objectives

At the end of this lab you will be able to:

- Test limits written to the NGINX access and error log files.

Steps

6. In another terminal, view request information by using the tail -f command on the access log.

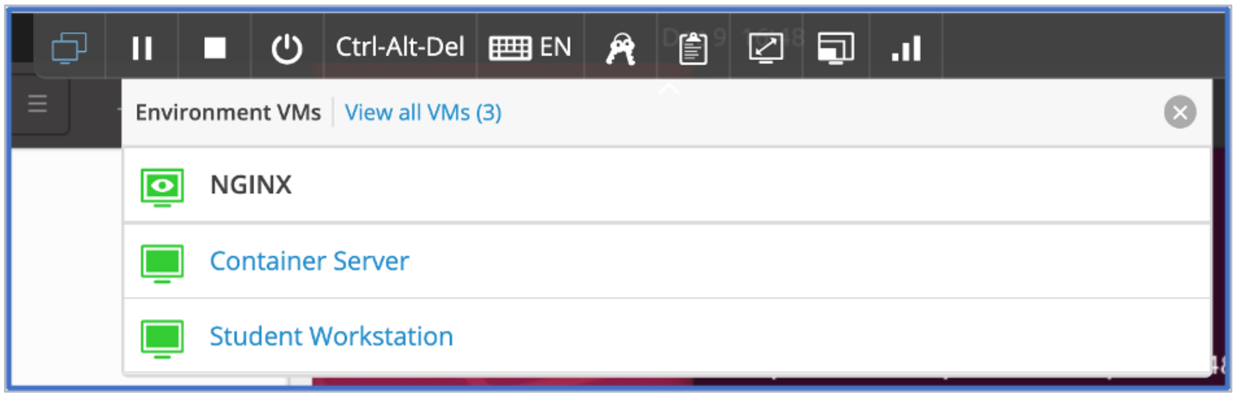
```
$ sudo tail -f /var/log/nginx/juice.access.log
```

> NOTE
> When viewing log files, it is best to have a wide but not too tall terminal.

7. In another terminal, view the error information by using the tail -f command with the error log file.

```
$ sudo tail -f /var/log/nginx/juice.error.log
```

8. At the top of the Skytap environment, click on the two monitors so you can select the **Student Workstation** lab system.

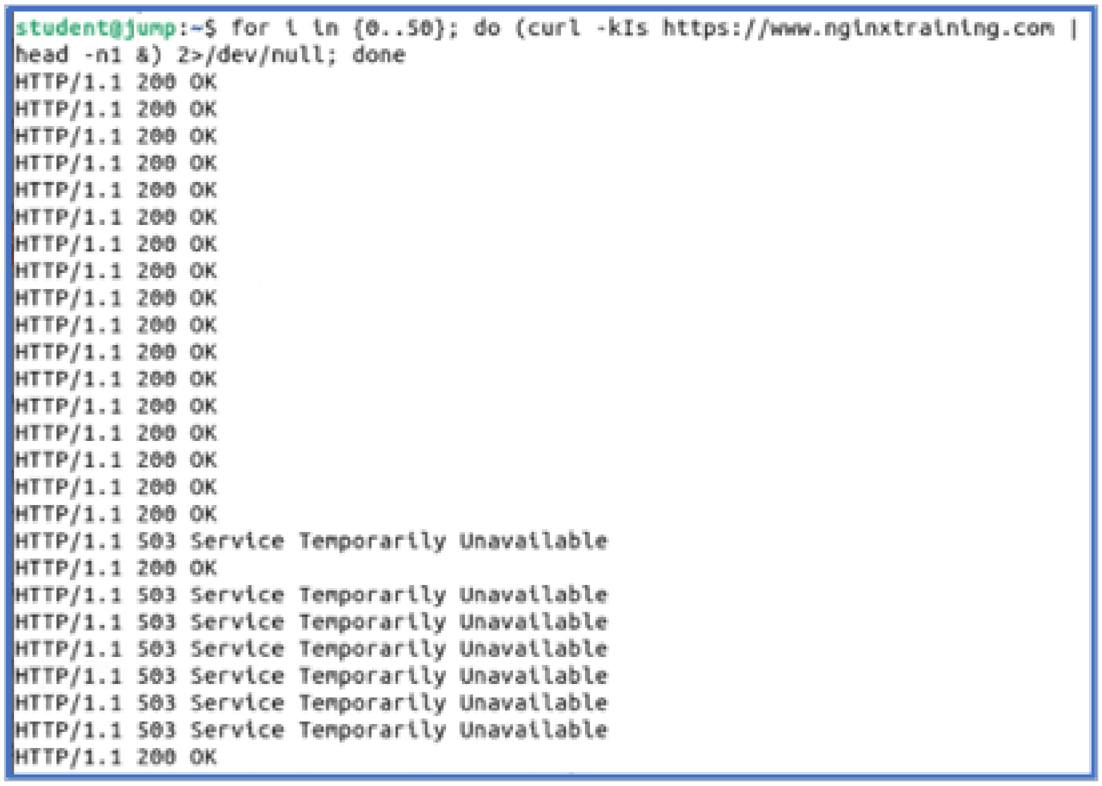


9. Log in to the **Student Workstation** lab system, using **student** for the password.

10. On the Student Workstation lab system, open a terminal and run this **for** loop command by copying from the SNIPPETS.txt file.

    ```
    $ for i in {0..50}; do (curl -kIs https://www.nginxtraining.com | head -n1
    &) 2> /dev/null; done
    ```

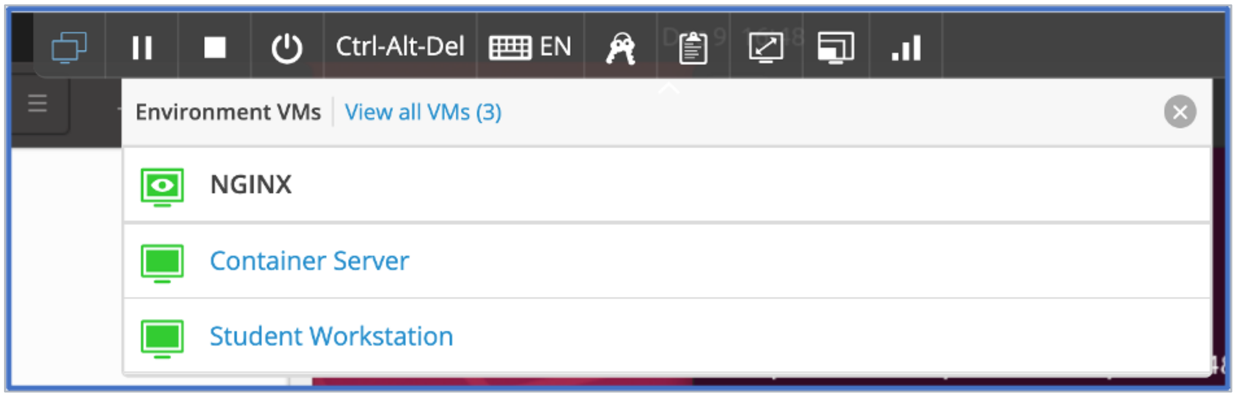
11. View the output from the for loop on your **Student Workstation** lab system:



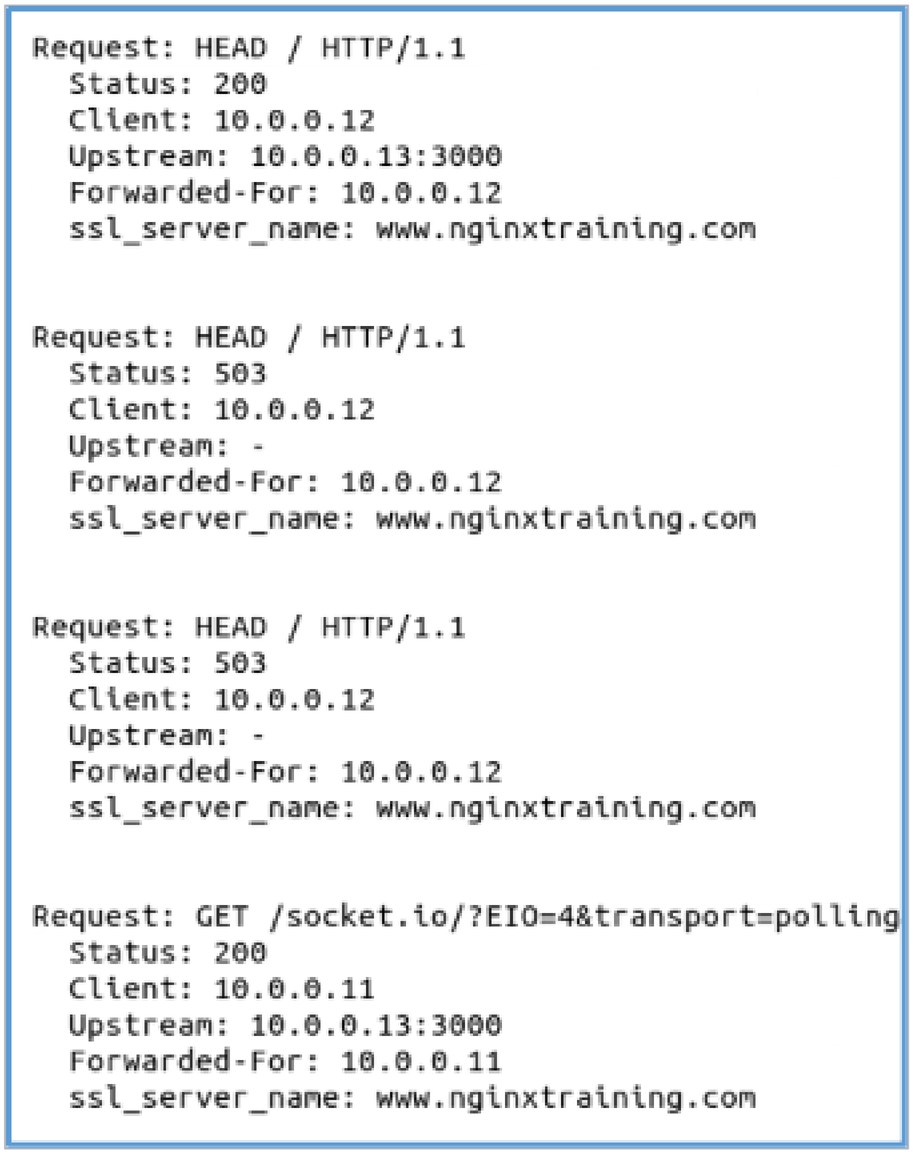
```
student@jump:~$ for i in {0..50}; do (curl -kIs https://www.nginxtraining.com |
head -n1 &) 2>/dev/null; done
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 200 OK
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 503 Service Temporarily Unavailable
HTTP/1.1 200 OK
```

NOTE
The for loop sends more requests than what the limits are set to on your NGINX lab system (10.0.0.11) resulting in several ***Service Temporarily Unavailable*** messages.

12. At the top of the Skytap environment, click on the two monitors so you can select the **NGINX** lab system.



13. View the access log output from the **tail** command on your **NGINX** lab system.

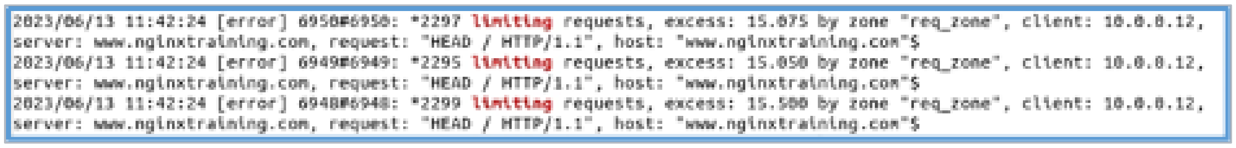


```
Request: HEAD / HTTP/1.1
  Status: 200
  Client: 10.0.0.12
  Upstream: 10.0.0.13:3000
  Forwarded-For: 10.0.0.12
  ssl_server_name: www.nginxtraining.com


Request: HEAD / HTTP/1.1
  Status: 503
  Client: 10.0.0.12
  Upstream: -
  Forwarded-For: 10.0.0.12
  ssl_server_name: www.nginxtraining.com


Request: HEAD / HTTP/1.1
  Status: 503
  Client: 10.0.0.12
  Upstream: -
  Forwarded-For: 10.0.0.12
  ssl_server_name: www.nginxtraining.com


Request: GET /socket.io/?EIO=4&transport=polling
  Status: 200
  Client: 10.0.0.11
  Upstream: 10.0.0.13:3000
  Forwarded-For: 10.0.0.11
  ssl_server_name: www.nginxtraining.com
```

NOTE
You should get several **503** messages once the limit has been reached. The **Status: 200** indicates the request was successful. **Client: 10.0.0.12** indicates where the request originated, which is your Student Workstation lab system, where the for loop command was run. **Upstream: 10.0.0.13:3000** indicates where NGINX forwarded the request to the Server Container lab system where the Juice Shop application is running on port 3000.

14. Use the following command to search thru the juice.error.log file on your NGINX lab system to view several rate limit warning messages about excess requests.

```
$ sudo cat -vet /var/log/nginx/juice.error.log | grep limiting
```



```
2023/06/13 11:42:24 [error] 6950#6950: *2297 limiting requests, excess: 15.075 by zone "req_zone", client: 10.0.0.12,
server: www.nginxtraining.com, request: "HEAD / HTTP/1.1", host: "www.nginxtraining.com"$
2023/06/13 11:42:24 [error] 6949#6949: *2295 limiting requests, excess: 15.050 by zone "req_zone", client: 10.0.0.12,
server: www.nginxtraining.com, request: "HEAD / HTTP/1.1", host: "www.nginxtraining.com"$
2023/06/13 11:42:24 [error] 6948#6948: *2299 limiting requests, excess: 15.500 by zone "req_zone", client: 10.0.0.12,
server: www.nginxtraining.com, request: "HEAD / HTTP/1.1", host: "www.nginxtraining.com"$
```

NOTE
You should get several **503** messages once the limit has been reached. The **Status: 200** indicates the request was successful. **Client: 10.0.0.12** indicates where the request originated, which is your Student Workstation lab system, where the for loop command was run. **Upstream: 10.0.0.13:3000** indicates where NGINX forwarded the request to the Server Container lab system where the Juice Shop application is running on port 3000.

Expected Results

After configuring rate limits in NGINX, you will have several errors and warning messages from sending more requests than allowed.

Lab 2: Logging Limit Rates

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- NGINX configuration file set up with rate limits.

Objectives

At the end of this lab you will be able to:

- Configure the NGINX limit request log level.
- Configure the NGINX limit request status response code.

Steps

1. On the **NGINX** lab system, edit the juice.conf file.

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

2. In the ssl server context, add the **limit_req_log_level** and the **limit_req_status**. Set the log_level to warn when the error response code limit is exceeded.

```
limit_req_log_level warn;
limit_req_status 4444;
```

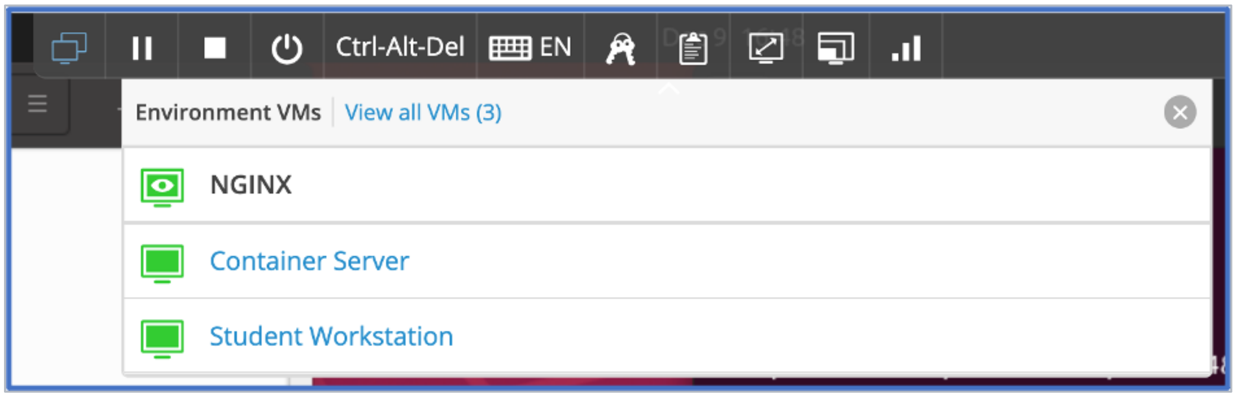
3. Save your configuration file and reload NGINX.

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

4. View the access_log file.

```
$ sudo tail -f /var/log/nginx/juice.access.log
```

5. At the top of the Skytap environment, click on the two monitors so you can select the **Student Workstation** lab system.

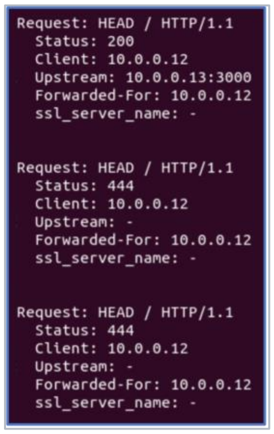


6. On the **Student Workstation** lab system, open a terminal and run this ***for*** loop command by copying from the SNIPPETS file.

```
$ for i in {0..50}; do (curl -kIs https://www.nginxtraining.com | head -n1
&) 2> /dev/null; done
```

7. View the output from the for loop on your Student Workstation lab system. There should only be HTTP/1.1 200 OK messages for the successful requests.

8. Switch back to your **NGINX** lab system to view the output from the NGINX access log. Note: You may have to scroll back to see the 444 status messages.



```
Request: HEAD / HTTP/1.1
  Status: 200
  Client: 10.0.0.12
  Upstream: 10.0.0.13:3000
  Forwarded-For: 10.0.0.12
  ssl_server_name: -


Request: HEAD / HTTP/1.1
  Status: 444
  Client: 10.0.0.12
  Upstream: -
  Forwarded-For: 10.0.0.12
  ssl_server_name: -


Request: HEAD / HTTP/1.1
  Status: 444
  Client: 10.0.0.12
  Upstream: -
  Forwarded-For: 10.0.0.12
  ssl_server_name: -
```

NOTE
The output from the access log has the response code 444 which was set in our configuration file for when the error response code limit is exceeded. The 444 status closes the TCP connection to NGINX.

TROUBLESHOOTING:
If you do not get any **Status: 444** messages in the NGINX access log file, make sure to scroll back in the log file or use this command:
grep 444 /var/log/nginx/juice.access.log

9. Use control c to exit the tail command.

10. For class purposes go back to your juice.conf file that does not have limits by renaming juice.conf to juice.LIMITS and then rename the juice.PRElimits file to juice.conf.

```
$ sudo mv /etc/nginx/conf.d/juice.{conf,LIMITS}
$ sudo mv /etc/nginx/conf.d/juice.{PRElimits,conf}
```

Expected Results

Configure and test logging limit rates by setting the limit request log level and setting the response status to 444 to terminate TCP connection.

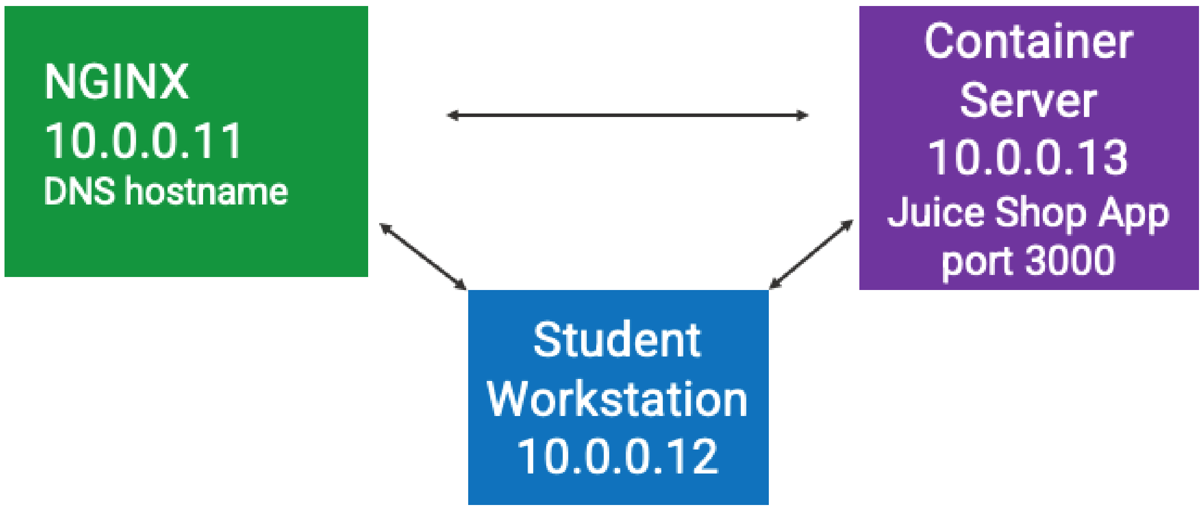
Labs - Setup Dynamic IP Deny-list

Lab Contents

1. **Create Dynamic IP Deny-list**
2. **Set the `$target`**
3. **Populate keyval Store**
4. **Test keyval_zone**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology



Lab 1: Dynamic IP Deny-List

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- On the NGINX lab system, configure juice.conf file to access the Juice Shop application with HTTPS.

Objectives

At the end of this lab you will be able to:

- Create a Dynamic IP Deny-list
- Use the NGINX Plus keyval store

Steps

11. Create a directory that the nginx user has write access to store the key value file.

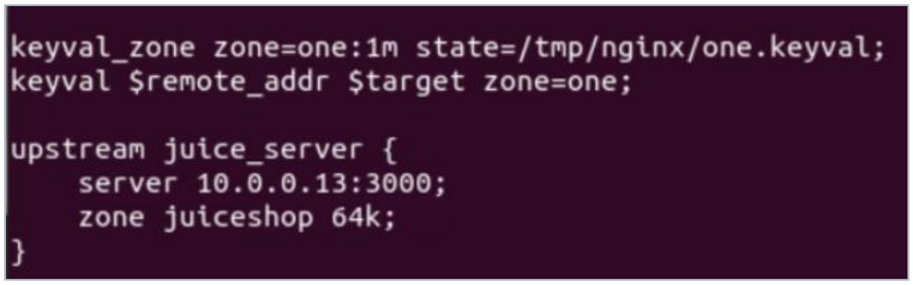
```
$ sudo mkdir /tmp/nginx
$ sudo chown nginx:nginx /tmp/nginx
```

12. Edit the juice.conf configuration file.

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

13. In the http context add a **keyval_zone** with the **state** parameter and add a keyval data store, mapping the client's ip address ($remote_addr) to the $target variable.

```
keyval_zone zone=one:1m state=/tmp/nginx/one.keyval;
keyval $remote_addr $target zone=one;
```



```
keyval_zone zone=one:1m state=/tmp/nginx/one.keyval;
keyval $remote_addr $target zone=one;

upstream juice_server {
    server 10.0.0.13:3000;
    zone juiceshop 64k;
}
```

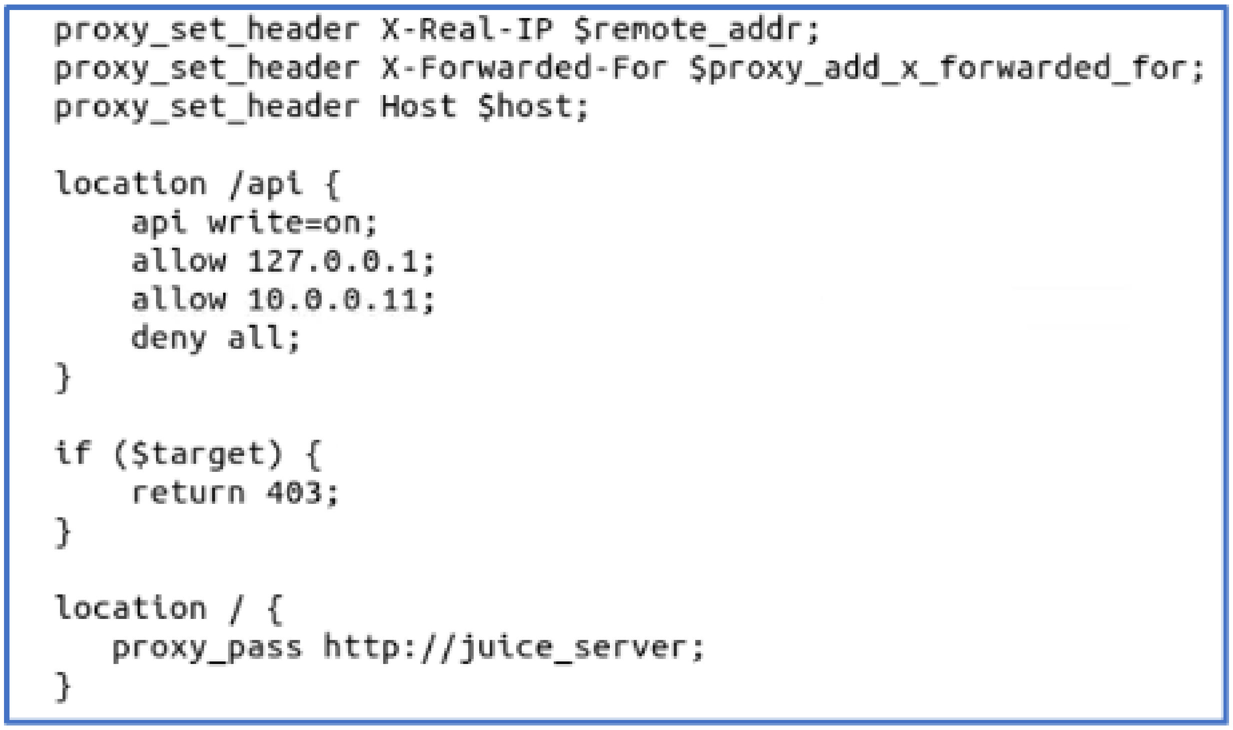
14. Continuing editing the juice.conf file, to create a new /api prefix in server 443 with write mode enabled on localhost only.

```
location /api {
    api write=on;
    allow 127.0.0.1;
    deny all;
}
```

15. Continuing editing the juice.conf file and after the api location add an **if** directive in the ssl server context with $target as the argument and a return status 403.

```
if ($target) {
    return 403;
}
```

NOTE
All of the changes you made to the juice.conf file are shown below (but this is not the whole file):



```
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header Host $host;

location /api {
    api write=on;
    allow 127.0.0.1;
    allow 10.0.0.11;
    deny all;
}

if ($target) {
    return 403;
}

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

16. Save your configuration file and reload NGINX.

```
$ sudo nginx -t && sudo nginx -s reload
```

17. Send a POST request to populate the keyval database with your **Student Workstation** lab systems IP address (10.0.0.12). By setting a value of “1” the Student Workstation lab system will be on the deny-list. You should get a **201 Created** response code which means your curl command was successful. Remember to copy the curl command from the SNIPPETS text file.

```
$ curl -k -v -H "Content-Type: application/json" -d ‘{"10.0.0.12":"1"}’
‘https://127.0.0.1/api/8/http/keyvals/one’
```

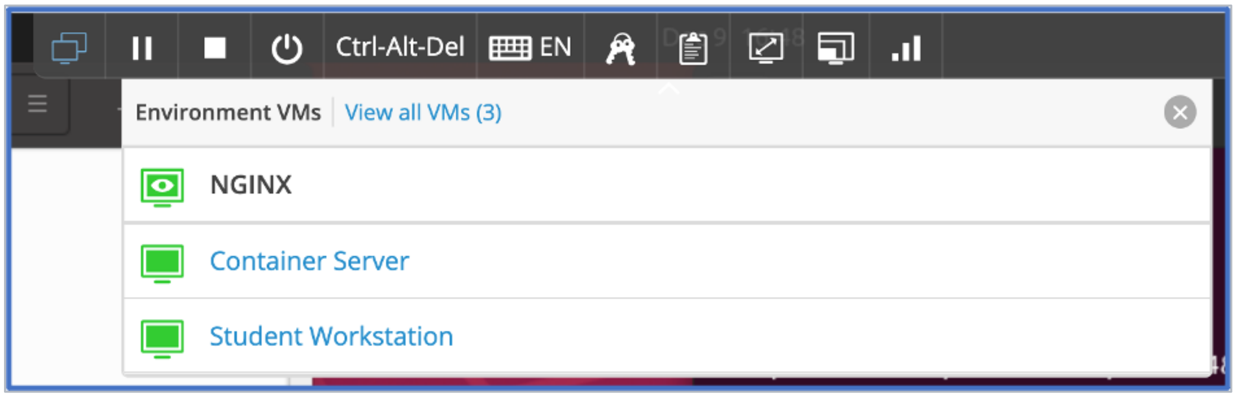
NOTE
You should get a “201 Created” response code, which means your curl command worked.

TROUBLESHOOTING:
If you get a 409 Conflict message showing the KeyvalKeyExists then you already have a value in memory (test with the next lab step) and you can continue or you could clear out the data and do this command again.

18. Send a GET request to confirm it works. Your output should be: {"10.0.0.12":"1"}

```
$ curl -k https://127.0.0.1/api/8/http/keyvals/one
```

19. At the top of the Skytap environment, click on the two monitors so you can select the **Student Workstation** lab system.



20. On the **Student Workstation** lab system, open a browser and enter your **NGINX** lab systems IP address 10.0.0.11. You should get a “**403 Forbidden**” response code

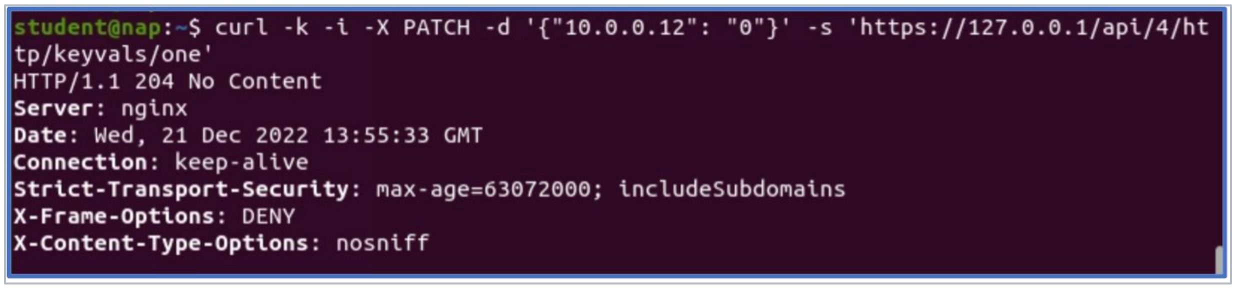
since the data part of the key-value pair is set to “1” which means “true” so NGINX will deny your Student Workstation lab system since it is on the deny-list.

https://www.nginxtraining.com

TROUBLESHOOTING:
You may have to click on some browser security questions to get to the 403 Forbidden response code: clear your browser data cache or use an incognito or private browser window.

21. Switch back to the **NGINX** lab system to use the NGINX api to update the **keyval_zone** data to change the value to a zero. You should get a "204 No Content" response.

```
$ curl -k -i -X PATCH -d ‘{“10.0.0.12”:”0”}’ -s
‘https://127.0.0.1/api/8/http/keyvals/one’
```



```
student@nap:~$ curl -k -i -X PATCH -d '{"10.0.0.12": "0"}' -s 'https://127.0.0.1/api/4/ht
tp/keyvals/one'
HTTP/1.1 204 No Content
Server: nginx
Date: Wed, 21 Dec 2022 13:55:33 GMT
Connection: keep-alive
Strict-Transport-Security: max-age=63072000; includeSubdomains
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
```

22. On the NGINX lab system, send a GET request to confirm it updated the value from “1” to “0”.

```
$ curl -k https://127.0.0.1/api/8/http/keyvals/one
```

Your output should be:
```
{"10.0.0.12":"0"}
```

23. Switch back to the **Student Workstation** lab system and in the browser, refresh the webpage https://10.0.0.11 to view if the result is different than previously?

NOTE
Yes. You should now be able to get to the Juice Shop web page without a Forbidden error message because the “0” means false, i.e., don’t denylist things from this IP address.

24. Switch back to the NGINX lab system and use the NGINX api to update the **keyval_zone** data to **DELETE** the entry. You should get a 204 No Content response.

```
$ curl -k -i -X DELETE -d ‘{“10.0.0.12”:”0”}’ -s
‘https://127.0.0.1/api/8/http/keyvals/one’
```

25. Verify that the value has been deleted.

```
$ curl -k https://127.0.0.1/api/4/http/keyvals/one
```

Your output should be:

```
{}
```

Expected Results

You have tested using a deny-list in your NGINX configuration.

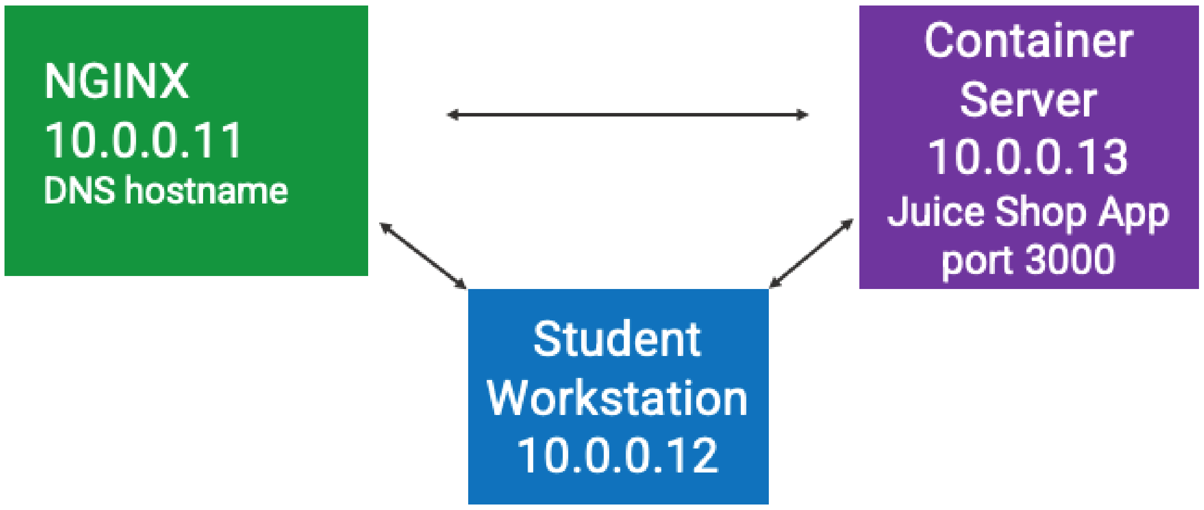
Labs - Dynamic Lazy Certificates

Lab Contents

1. **Use Dynamic Lazy Certificate Loading from disk**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology



Lab 1: Use Dynamic Lazy Certificates from disk

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- On your NGINX lab system a juice.conf file with an upstream to the Juice Shop application with certs and HTTPS set up.

Objectives

At the end of this lab you will be able to:

- Use NGINX Dynamic Lazy Certificate file loading from disk.

Steps

1. Before changing to use the NGINX Dynamic Certificate loading method, backup the configuration files: ssl-params.conf, juice.conf, and api_server.conf

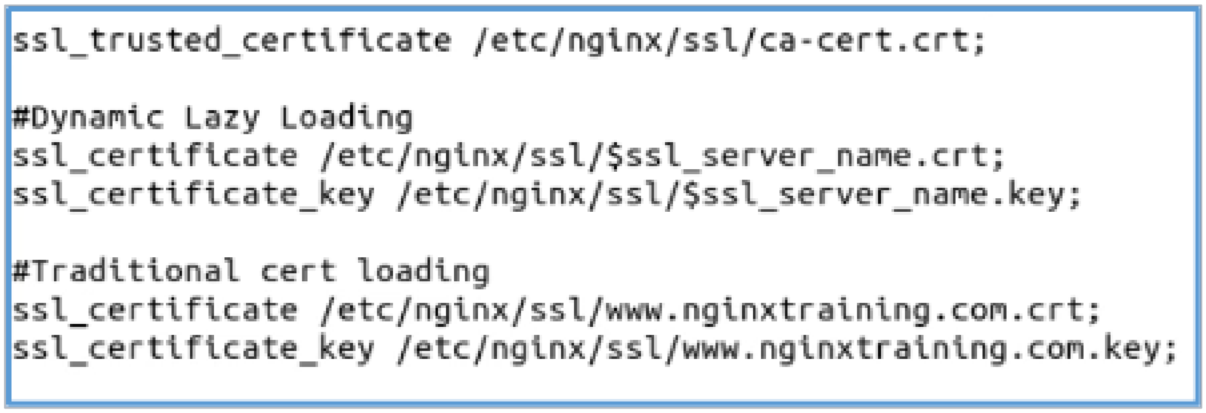
```
$ sudo cp /etc/nginx/ssl-configs/ssl-params.{conf,traditional}
$ sudo cp /etc/nginx/conf.d/juice.{conf,traditional}
$ sudo cp /etc/nginx/conf.d/api_server.{conf,traditional}
```

2. Edit the **ssl-params.conf** file to add new ssl certificate directives to use Dynamic Lazy loading. These directives use special variables for dynamically loading the appropriate certificate based on SNI.

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

ssl_certificate /etc/nginx/ssl-configs/certs/$ssl_server_name.crt;
ssl_certificate_key /etc/nginx/ssl-configs/certs/$ssl_server_name.key;
```

NOTE
The ssl_trusted_certificate can’t be loaded dynamically.

3. Comment out the original 2 certificate directives that use the traditional method of listing each certificate file and pathname on disk. The beginning of the ssl-params.conf file is shown.



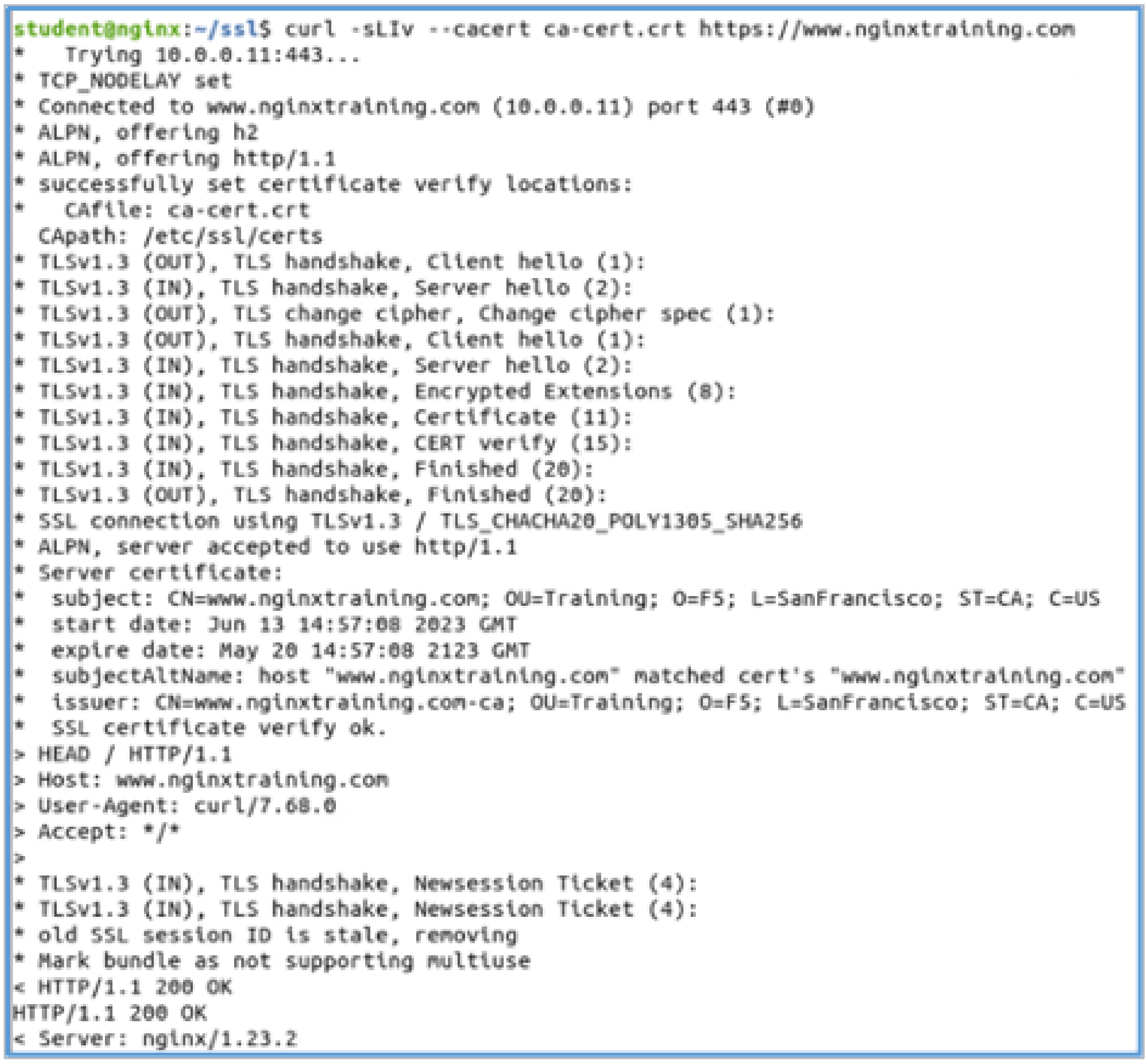
```
ssl_trusted_certificate /etc/nginx/ssl/ca-cert.crt;

#Dynamic Lazy Loading
ssl_certificate /etc/nginx/ssl/$ssl_server_name.crt;
ssl_certificate_key /etc/nginx/ssl/$ssl_server_name.key;

#Traditional cert loading
ssl_certificate /etc/nginx/ssl/www.nginxtraining.com.crt;
ssl_certificate_key /etc/nginx/ssl/www.nginxtraining.com.key;
```

4. Test and reload NGINX to use your updated configuration files.

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

5. Test your configuration with curl. Make sure to scroll back to the beginning output to see the connection, port information and response 200 OK.

```
$ cd ~/ssl
$ curl -sLIvk --cacert ca-cert.crt https://www.nginxtraining.com
```



```
student@nginx:~/ssl$ curl -sLIv --cacert ca-cert.crt https://www.nginxtraining.com
*   Trying 10.0.0.11:443...
* TCP_NODELAY set
* Connected to www.nginxtraining.com (10.0.0.11) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: ca-cert.crt
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_CHACHA20_POLY1305_SHA256
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: CN=www.nginxtraining.com; OU=Training; O=F5; L=SanFrancisco; ST=CA; C=US
*  start date: Jun 13 14:57:08 2023 GMT
*  expire date: May 20 14:57:08 2123 GMT
*  subjectAltName: host "www.nginxtraining.com" matched cert's "www.nginxtraining.com"
*  issuer: CN=www.nginxtraining.com-ca; OU=Training; O=F5; L=SanFrancisco; ST=CA; C=US
*  SSL certificate verify ok.
> HEAD / HTTP/1.1
> Host: www.nginxtraining.com
> User-Agent: curl/7.68.0
> Accept: */*
>
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Server: nginx/1.23.2
```

Expected Results

You have configured NGINX to use dynamic lazy certificate loading instead of the traditional certificate loading method.

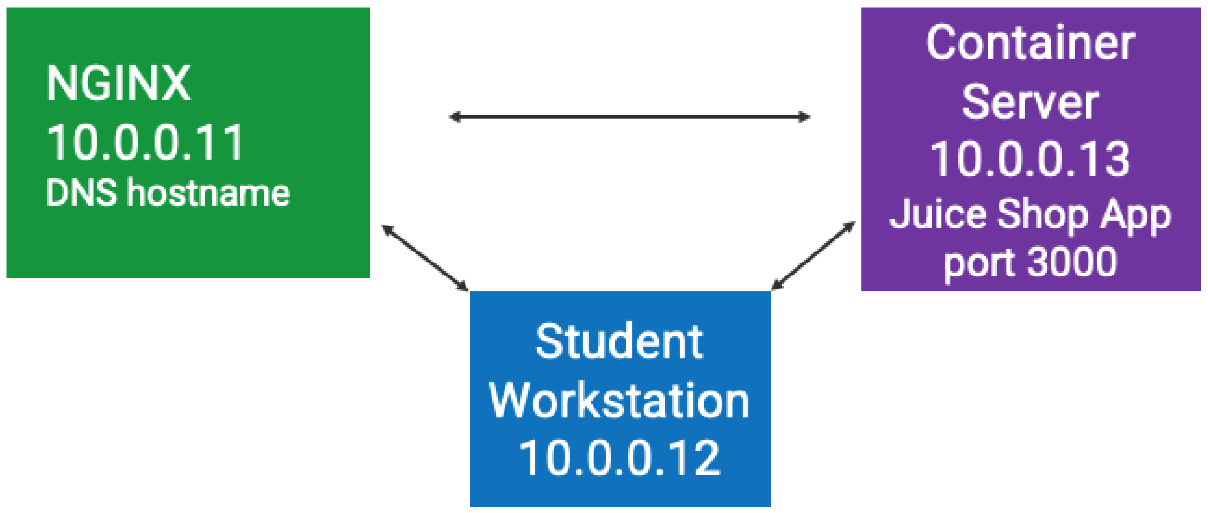
Labs - Dynamic In-Memory Certificate

Lab Contents

1. **Using NGINX Dynamic In-memory Certificates**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology



Lab 1: Use Dynamic In-memory Certificate

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- On your NGINX lab system a juice.conf file with an upstream to the Juice Shop application.

Objectives

At the end of this lab you will be able to:

- Configure and test using NGINX Dynamic In-memory certificates.

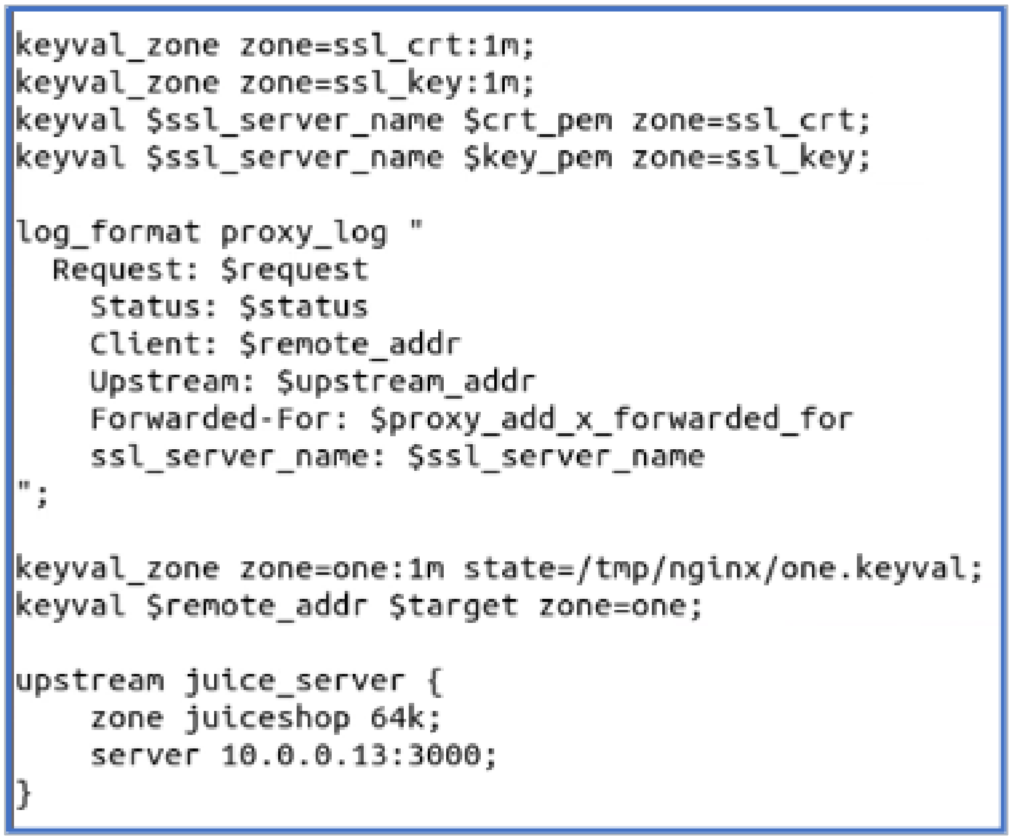
Steps

1. Before changing to use the NGINX Dynamic In-memory certificate loading method, backup the following configuration files: **ssl-params.conf**, **api_server.conf**, and **juice.conf**.

```
$ sudo cp /etc/nginx/ssl-configs/ssl-params.{conf,lazy}
$ sudo cp /etc/nginx/conf.d/api_server.{conf,lazy}
$ sudo cp /etc/nginx/conf.d/juice.{conf,lazy}
```

2. Edit the **juice.conf** file in the http context (top of file) add two new **keyval_zone** directives; one for the certificate data and one for the private key data and then add two new **keyval** directives mapping the key values and zones. Save the juice.conf file.

```
keyval_zone zone=ssl_crt:1m;
keyval_zone zone=ssl_key:1m;
keyval $ssl_server_name $crt_pem zone=ssl_crt;
keyval $ssl_server_name $key_pem zone=ssl_key;
```



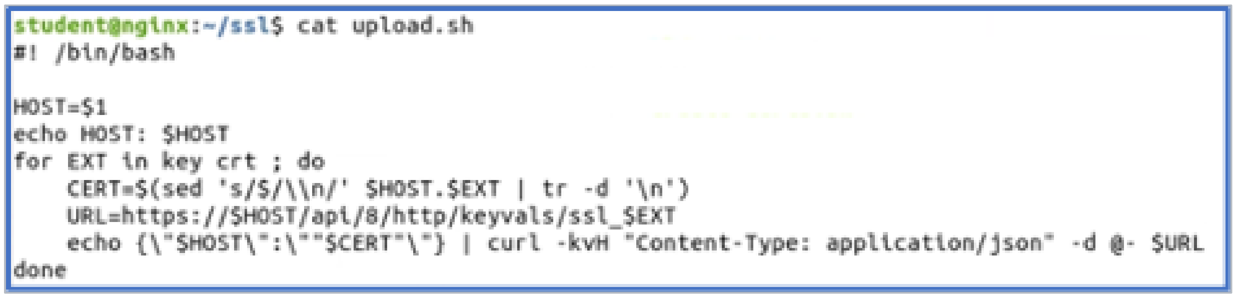
```
keyval_zone zone=ssl_crt:1m;
keyval_zone zone=ssl_key:1m;
keyval $ssl_server_name $crt_pem zone=ssl_crt;
keyval $ssl_server_name $key_pem zone=ssl_key;

log_format proxy_log "
  Request: $request
    Status: $status
    Client: $remote_addr
    Upstream: $upstream_addr
    Forwarded-For: $proxy_add_x_forwarded_for
    ssl_server_name: $ssl_server_name
";

keyval_zone zone=one:1m state=/tmp/nginx/one.keyval;
keyval $remote_addr $target zone=one;

upstream juice_server {
    zone juiceshop 64k;
    server 10.0.0.13:3000;
}
```

3. Reload NGINX to use your updated configuration files

```
$ sudo nginx -t && sudo nginx -s reload
```

4. Configure the certificate and key files into a format that NGINX can store in memory and that your student account has access to. View the **upload.sh** script file.

```
$ cd ~/ssl
$ cat upload.sh
```



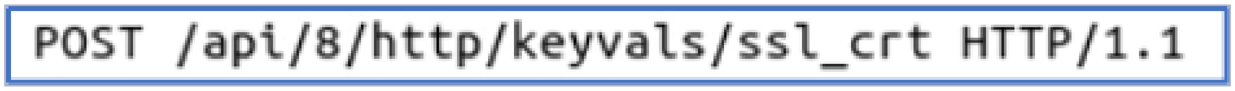
```
student@nginx:~/ssl$ cat upload.sh
#! /bin/bash

HOST=$1
echo HOST: $HOST
for EXT in key crt ; do
    CERT=$(sed 's/$/\\n/' $HOST.$EXT | tr -d '\n')
    URL=https://$HOST/api/8/http/keyvals/ssl_$EXT
    echo {\"$HOST\":\""$CERT"\"} | curl -kvH "Content-Type: application/json" -d @- $URL
done
```

5. Run the script **upload.sh** passing it the DNS name of your NGINX system. You must run this script while in the ~/ssl directory so it has access to the cert files.

```
$ ./upload.sh www.nginxtraining.com
```

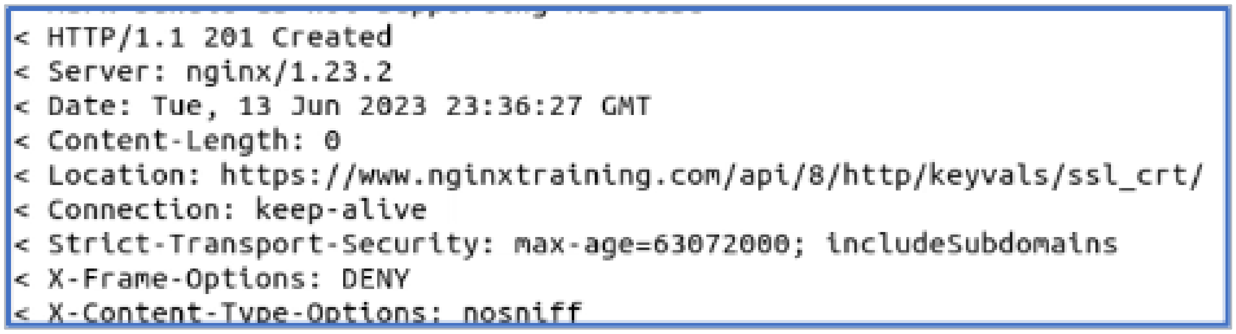
There should be 2 of these “201 Created” lines - one for the key file and one for the cert file.

If you look closely at the output there is a **POST** HTTP request which shows where the value is being written.



```
POST /api/8/http/keyvals/ssl_crt HTTP/1.1
```

Also, the **Location** line contains the URL to the written memory file which you can retrieve using **curl -k Location_URL_value**.



```
< HTTP/1.1 201 Created
< Server: nginx/1.23.2
< Date: Tue, 13 Jun 2023 23:36:27 GMT
< Content-Length: 0
< Location: https://www.nginxtraining.com/api/8/http/keyvals/ssl_crt/
< Connection: keep-alive
< Strict-Transport-Security: max-age=63072000; includeSubdomains
< X-Frame-Options: DENY
< X-Content-Type-Options: nosniff
```

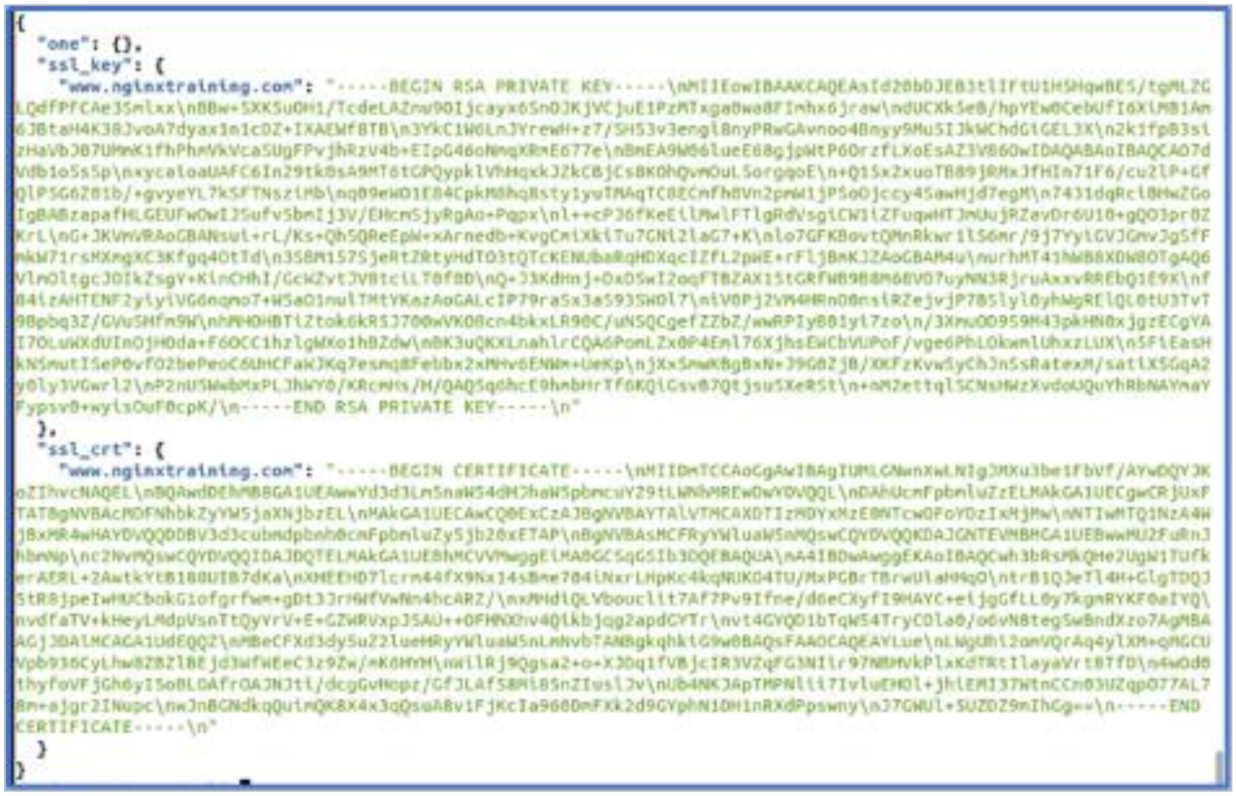
Troubleshooting:
Make sure you runt the UPLOAD script in the terminal where you have the nginx group as part of your student account (where you ran exec su -l student) and make sure you are in the directory /etc/nginx/ssl-configs/certs.

If you need to delete the key value store memory entry and run the UPLOAD script again do these commands:

```
$ curl -k -i -X DELETE https://$DNS/api/6/http/keyvals/ssl_key
$ curl -k -i -X DELETE https://$DNS/api/6/http/keyvals/ssl_crt

OR

$ sudo systemctl stop nginx
$ sudo systemctl status nginx
$ sudo systemctl start nginx
```

6. To test that both the certificate and private key files are in memory in the keyval store location.

```
$ curl -k https://www.nginxtraining.com/api/8/http/keyvals | jq
```

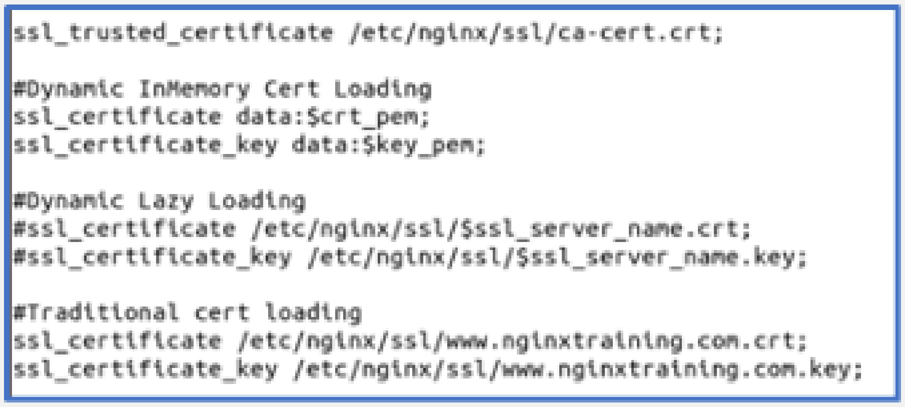


```
{
  "one": {},
  "ssl_key": {
    "www.nginxtraining.com": "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIBAAKCAQEAsId20bDJEB3tlIFtU1HSHqwBE5/tgMLZG
LQdfPfCAe3Smlxx\nBBw+SXK5uOH1/TcdeLAZnu9OIjcayx6SnOJKjVCjuE1PzMTxga8wa8FInhx6jraw\ndUCXk5e8/hpYEw0CebUfI6XlMB1Am
6JBtaH4K38JvoA7dyax1n1cDZ+IXAEWfBTB\n3YkC1W6LnJYrewH+z7/SH53v3engl8nyPRwGAvnoo4Bnyy9MuSIJkWChdG1GEL3X\n2k1fpB3si
zHaVbJB7UMmK1fhPhmVkVcaSUgFPvjhRzV4b+EIpG46oNmqXRmE677e\nBmEA9W06lueE68gjpWtP6OrzfLXoEsAZ3V860wIDAQABAoIBAQCAO7d
Vdb1o5s5p\nxycaloaUAFC6In29tk0sA9MT6tGPQypklVhHqxkJZkCBjCsBKOhQvmOuLSorgqoE\n+Q15x2xuoTB89jRMxJfHIn71F6/cu2lP+Gf
QlP5G6Z81b/+gvyeYL7kSFTNsziMb\nq09eWD1E84CpkM0hqBsty1yuTMAqTC8ECmfh8Vn2pmW1jP5oOjccy45awMjd7egM\n7431dqRcl8MwZGo
IgBABzapafHLGEUFwOwIJSufvSbmIj3V/EHcmSjyRgAo+Pqpx\nl++cP36fKeEilMwlFTlgRdVsgiCW1iZFuqwHTJmUujRZavDr6U10+gQO3pr0Z
KrL\nG+JKVmVRAoGBANsul+rL/Ks+Qh5QReEpW+xArnedb+KvgCmiXkiTu7GNi2laG7+K\nlo7GFKBovtQMnRkwr1lS6mr/9j7YyiGVJGmvJg5fF
mkW71rsMXmgXC3Kfgq4OtTd\n35BM157SjeRtZRtyHdTO3tQTcKENUbaRqHDXqcIZfL2pwE+rFljBmKJZAoGBAM4u\nurhMT41NWB8XDW8OTgAQ6
VlmOltgcJOIkZsgY+KinCHhI/GcWZvtJVBtciLT0f0D\nQ+J3KdHnj+OxO5wI2oqFTBZAX15tGRfWB9B8M68VO7uyNN3RjruAxxvRREbQ1E9X\nf
B4izAHTENF2yiyiVG6nqmo7+W5aO1nulTMtYKazAoGALcIP79ra5x3a593SHOl7\niV0Pj2VM4HRnO0nsiRZejvjP7BSlyl0yhWgRElQL0tU3TvT
9Bpbq3Z/GVuSHfm9W\nhMHOHBTiZtok6kRSJ700wVKO8cn4bkxLR90C/uNSQCgefZZbZ/wwRPIyB01yi7zo\n/3XmuOD959M43pkHN0xjgzECgYA
I7OLuWXdUInOjHOda+F6OCC1hzlgWXo1hBZdw\nBK3uQKXLnahlrCQA6PomLZx0P4Eml76XjhsEWCbVUPoF/vge6PhLOkwnlUhxzLUX\n5FiEasH
kNSmutISeP0vfO2bePeoC6UHCFaKJKq7esnq8Febbx2xMHv6ENWm+UeKp\njXx5mwKBgBxN+J9G8ZjB/XKFzKvw5yChJnSsRatexM/satiX5GqA2
y0ly3VGwrl2\nP2nU5WwbMxPLJhWY0/KRcmHs/M/QAQ5q0hcE9hmbHrTf6KQiGsv87QtjsuSXeRSt\n+nMZettqlSCNsMWzXvdoUQuYhRbNAYmaY
Fypsv0+wyisOuF0cpK/\n-----END RSA PRIVATE KEY-----\n"
  },
  "ssl_crt": {
    "www.nginxtraining.com": "-----BEGIN CERTIFICATE-----\nMIIDmTCCAoGgAwIBAgIUMLGNwnXwLNIgJMXu3be1fbVf/AYwDQYJK
oZIhvcNAQEL\nBQAwdDELMBBGA1UEAwwYd3d3Lm5naW54dHJhaW5pbmcuY29tLWNhMREwDwYDVQQL\nDAhUcmFpbmluZzELMAkGA1UECgwCRjUxF
TATBgNVBAcMDFNhbkZyYW5jaXNjbzEL\nMAkGA1UECAwCQ0ExCzAJBgNVBAYTAlVTMCAXDTIzMDYxMzE0NTcwOFoYDzIxMjMw\nNTIwMTQ1NzA4W
jBxMR4wHAYDVQQDDBV3d3cubmdpbnh0cmFpbmluZy5jb20xETAP\nBgNVBAsMCFRyYWluaW5nMQswCQYDVQQKDAJGNTEVMBMGA1UEBwwMU2FuRnJ
hbmNp\nc2NvMQswCQYDVQQIDAJDQTELMAkGA1UEBhMCVVMwggEiMA0GCSqGSIb3DQEBAQUA\nA4IBDwAwggEKAoIBAQCwh3bRsMkQHe2UgW1Tufk
erAERL+2AwtkYtB1B8UIB7dKa\nXMEEHD7lcrm44fX9Nx14sBme704iNxrLHpKc4kqNUKO4TU/MxPGBrTBrwUlaHMqO\ntrB1Q3eTl4H+GlgTDQJ
StR8jpeIwHUCbokG1ofgrfwm+gDt3JrHWfVwNn4hcARZ/\nxMHdiQLVbouclit7Af7Pv9Ifne/d6eCXyfI9HAYC+eijgGfLL0y7kgmRYKF0aIYQ\
nvdfaTV+kHeyLMdpVsnTtQyVrV+E+GZWRVxpJ5AU++0FHNXhv4Qlkbjqg2apdCYTr\nvt4GYQD1bTqW54TryCOla0/o6vN8tegSwBndXzo7AgMBA
AGjJDAiMCAGA1UdEQQZ\nMBeCFXd3dy5uZ2lueHRyYWluaW5nLmNvbTANBgkqhkiG9w0BAQsFAAOCAQEAYLue\nLWgUhi2omVQrAq4ylXM+QMGCU
Vpb916CyLhw8ZBZlBEjd3WfWEeC3z9Zw/mK6MYM\nWilRj9Qgsa2+o+XJDq1fVBjcIR3VZqfG3NIir97NBMVkPlxKdTRtIlayaVrt8TfD\n4wOdB
thyfoVFjGh6yISoBLOAfrOAJNJti/dcgGvHopz/GfJLAf58Mi85nZIuslJv\nUb4NKJApTMPNlii7IvluEHOl+jhlEMI37WtnCCn03UZqpO77AL7
8m+ajgr2INupc\nwJnBGNdkqQuimQK8X4x3qQswA8v1FjKcIa960DmFXk2d9GYphN1DH1nRXdPpswny\nJ7GWUl+SUZDZ9mIhGg==\n-----END
CERTIFICATE-----\n"
  }
}
```

7. Now that the key and certificate are in our key-value memory, edit the file **ssl-params.conf** and **comment out** the existing lazy loading definitions for the **ssl_certificate** and **ssl_certificate_key** and then add the new method to use the Dynamic In-memory syntax **data:$variable.**

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

ssl_certificate data:$crt_pem;
ssl_certificate_key data:$key_pem;
```

NOTE
Your file should look like:



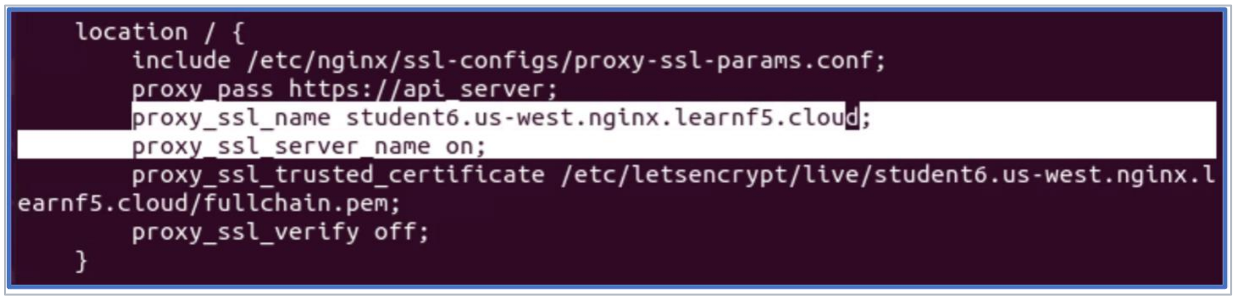
```
ssl_trusted_certificate /etc/nginx/ssl/ca-cert.crt;

#Dynamic InMemory Cert Loading
ssl_certificate data:$crt_pem;
ssl_certificate_key data:$key_pem;

#Dynamic Lazy Loading
#ssl_certificate /etc/nginx/ssl/$ssl_server_name.crt;
#ssl_certificate_key /etc/nginx/ssl/$ssl_server_name.key;

#Traditional cert loading
ssl_certificate /etc/nginx/ssl/www.nginxtraining.com.crt;
ssl_certificate_key /etc/nginx/ssl/www.nginxtraining.com.key;
```

NOTE: Leave the ssl_trusted_certificate directive as it is since it is not loaded dynamically.

8. Edit the api_server.conf file to add proxy ssl name information using the directives **proxy_ssl_name** and **proxy_ssl_server_name**. Add these to the 8080 server block under the location / and after your proxy_pass directive.

```
proxy_ssl_name www.nginxtraining.com;
proxy_ssl_server_name on;
```

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



```
    location / {
        include /etc/nginx/ssl-configs/proxy-ssl-params.conf;
        proxy_pass https://api_server;
        proxy_ssl_name student6.us-west.nginx.learnf5.cloud;
        proxy_ssl_server_name on;
        proxy_ssl_trusted_certificate /etc/letsencrypt/live/student6.us-west.nginx.l
earnf5.cloud/fullchain.pem;
        proxy_ssl_verify off;
    }
```

9. Test and reload NGINX to use your updated configuration files.

```
$ sudo nginx -t && sudo nginx -s reload
```

10. Send a GET request to confirm Dynamic Certificate loading is working. The output will be your private key and cert.

```
$ curl -k https://www.nginxtraining.com:8080
$ curl -k https://www.nginxtraining.com:8080
$ curl -k https://www.nginxtraining.com:8080
```

> NOTE
> You should get alternating upstreams responding such as “this is app1”, then “this is app2!” followed by “this is app1”.

11. Test access to the Juice Shop app.

```
$ curl -k https://www.nginxtraining.com
```

12. For our lab purposes revert back to using the traditional certificates by copying your backup files back to the original files. This will now use the traditional certificate.

```
$ sudo cp /etc/nginx/ssl-configs/ssl-params.{conf,dynamic-in-memory}
$ sudo cp /etc/nginx/ssl-configs/ssl-params.{traditional,conf}
$ sudo cp /etc/nginx/conf.d/juice.{conf,dynamic-in-memory}
$ sudo cp /etc/nginx/conf.d/api_server.{traditional,conf}
$ sudo cp /etc/nginx/conf.d/api_server.{conf,dynamic-in-memory}$ sudo cp
/etc/nginx/conf.d/juice.{traditional,conf}
```

13. Test and reload NGINX to use your updated configuration files.

```
$ sudo nginx -t && sudo nginx -s reload
```

14. Test your configuration with curl. Make sure to scroll back to the beginning output to see 200 OK message.

```
$ cd ~/ssl
$ curl -LIv --cacert ca-cert.crt https://www.nginxtraining.com
```

15. Test using a browser. Your output should be either “this is app1” or “this is app2” and then the Juice Shop app.

```
https://www.nginxtraining.com:8080
https://www.nginxtraining.com
```

Expected Results

You configure and test using the NGINX Dynamic In-memory certificates.

STOP

Chapter 5

Labs - Configure NGINX to use HTTP/2

Lab Contents

1. **Configure NGINX to use HTTP/2**
2. **Configure NGINX to not display the version number on error pages**

Estimated time to complete labs: **[XX] minutes**

Lab 1: Configure NGINX to use HTTP/2

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- Have the configuration file /etc/nginx/conf.d/api_server.conf

Objectives

At the end of this lab you will be able to:

- Configure NGINX to use HTTP/2.

Steps

16. Edit the **api_server.conf** file.

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

17. In the server context for 8080, add **http2** to the listen directive. Save the file.

    ```
    server {
        listen 8080 ssl http2;
    …
    }
    ```

18. Reload NGINX.

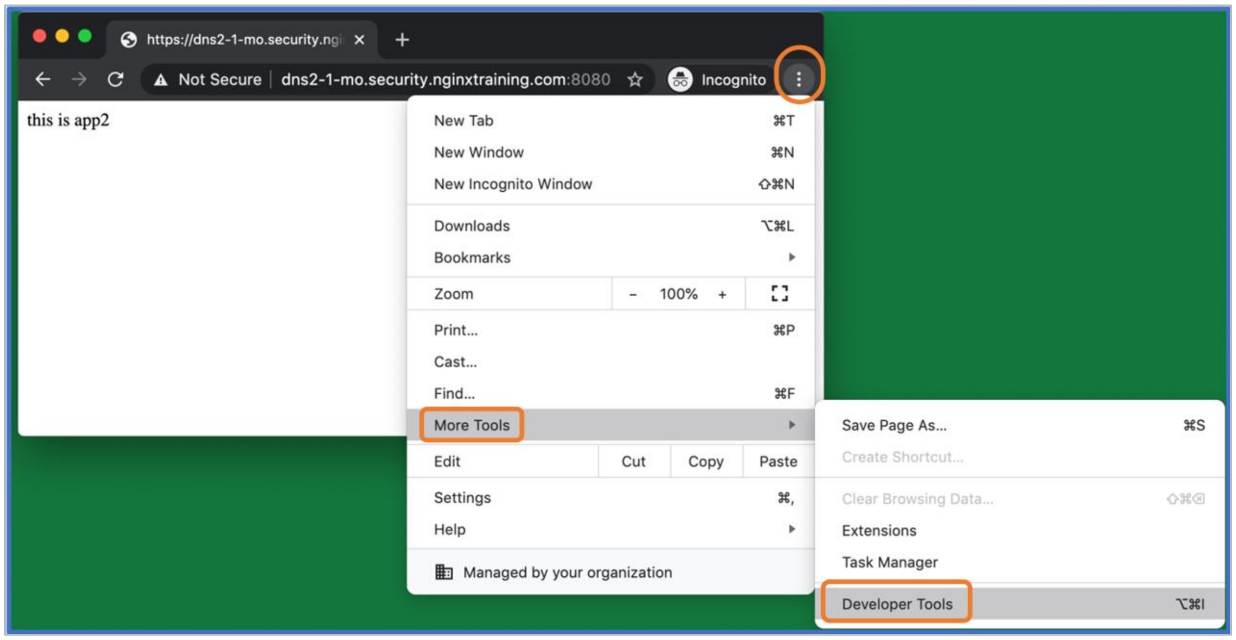
    ```
    $ sudo nginx -s reload
    ```

19. Test using a browser. Your output should be either “this is app1” or “this is app2”.

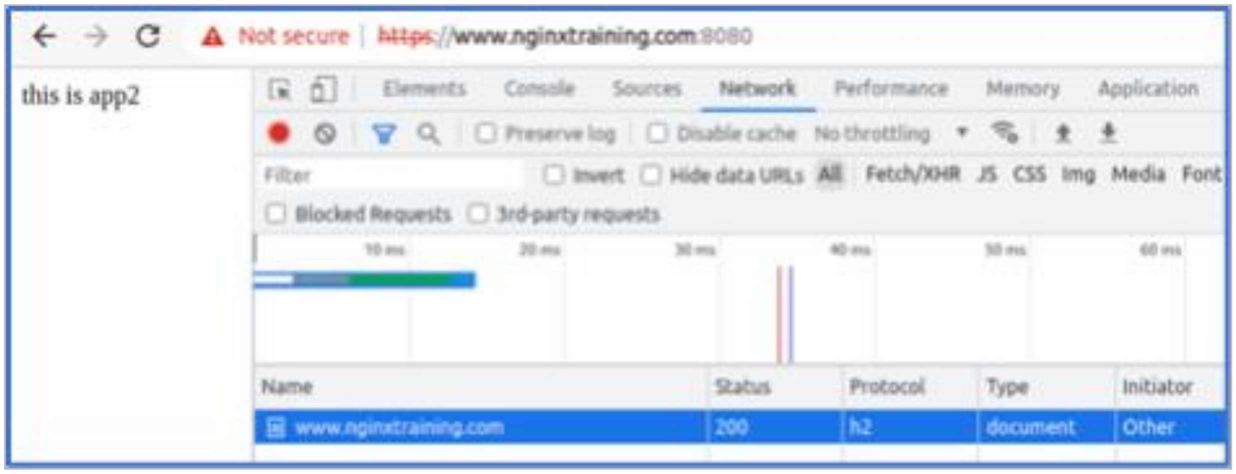
    ```
    https://<www.nginxtraining.com:8080
    ```

NOTE
If you use curl to test, it automatically converts HTTP/2 to HTTP1.1 for readability.

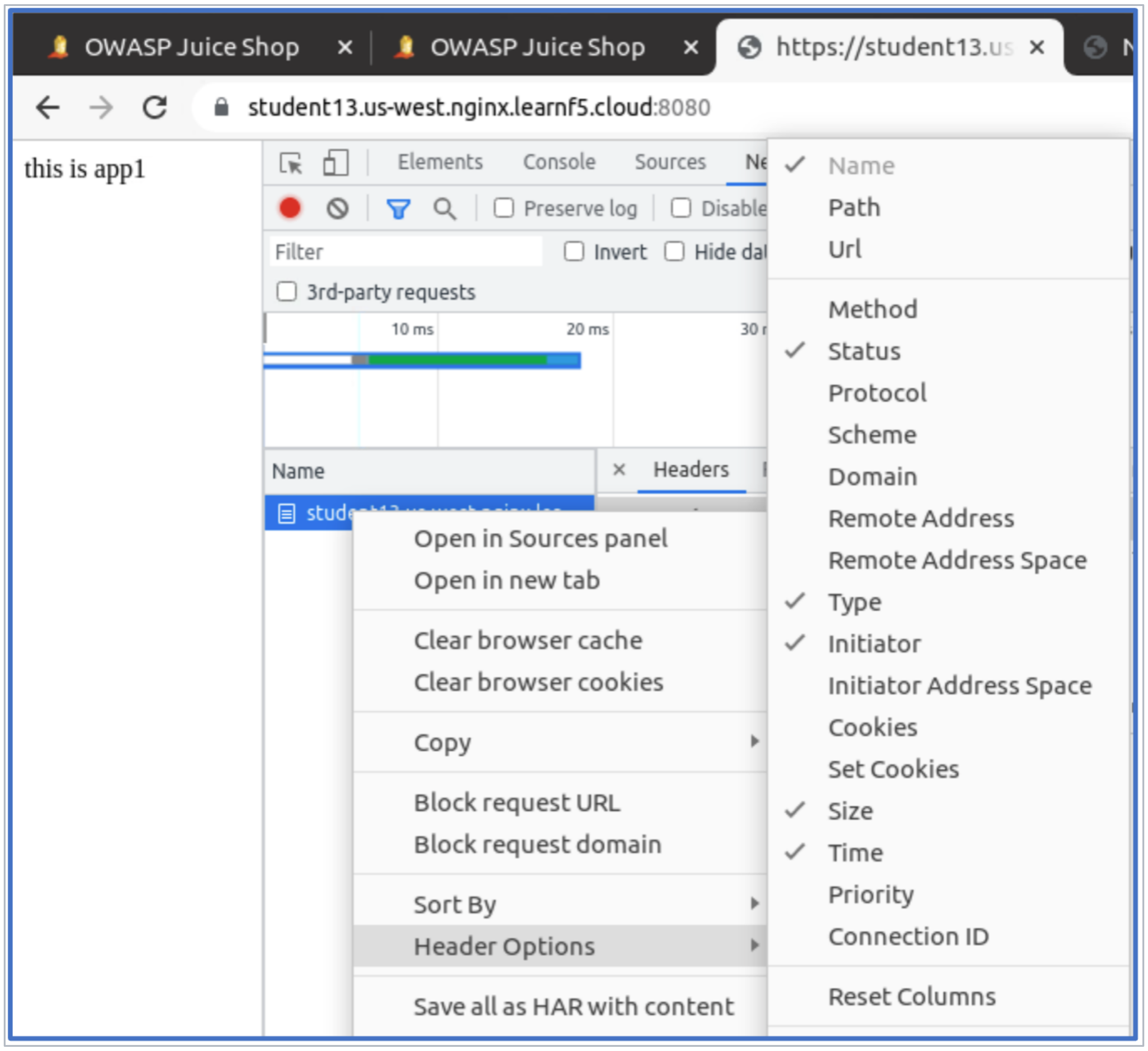
20. To verify that your request is using the http2 protocol, you will need to get further information from your browser. For the Chrome browser, click the **3 vertical dots**, select “**More Tools**” and then select “**Developer Tools**”.



21. Click the “**Network**” tab. Refresh the browser web page so that your DNS name is visible. The Status should be 200 and the Protocol **h2** for http2.


Not secure | https://www.nginxtraining.com:8080
this is app2
Elements Console Sources Network Performance Memory Application
Preserve log Disable cache No throttling
Filter Invert Hide data URLs All Fetch/XHR JS CSS Img Media Font
Blocked Requests 3rd-party requests

Name	Status	Protocol	Type	Initiator
www.nginxtraining.com	200	h2	document	Other



NOTE
If the Protocol column is not visible, with your lab systems name, www.nginxtraining.com, highlighted, right click to select it, choose Header Options and then click on Protocol and finally refresh the JuiceShop page. You might also want to close the console at the bottom of the screen if your systems name is not visible.

Expected Results

NGINX has now been configured to use the protocol HTTP/2.

Lab 2: Configure NGINX to not display version number on error pages

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- The configuration file /etc/nginx/conf.d/api_server.conf must exist.

Objectives

At the end of this lab you will be able to:

- Configure NGINX to not display the version number on error pages.

Steps

1. In a browser enter your DNS name with port 8080 to display a working webpage. (Close the developer tools.)

   ```
   https://www.nginxtraining.com:8080
   ```

2. In a browser enter your systems DNS name with port 8080 followed by the non-existent webpage “shop” so that the default NGINX behavior is displayed showing the version of NGINX.

   ```
   https://www.nginxtraining.com:8080/shop
   ```

3. Edit the **nginx.conf** file.

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

4. To not display the NGINX version number on error pages add the directive **server_tokens off** to the http context. Save the file.

   ```
   http {
      server_tokens off;
      include /etc/nginx/mime.types
      default_type application/octet-stream
   ```

```
…
}
```

5. Reload nginx.

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

6. In your browser, test that the NGINX version is no longer displayed when an error page is returned.

```
https://www.nginxtraining.com:8080/shop
```

Expected Results

NGINX is now configured to not display the version number on error pages.

Chapter 6

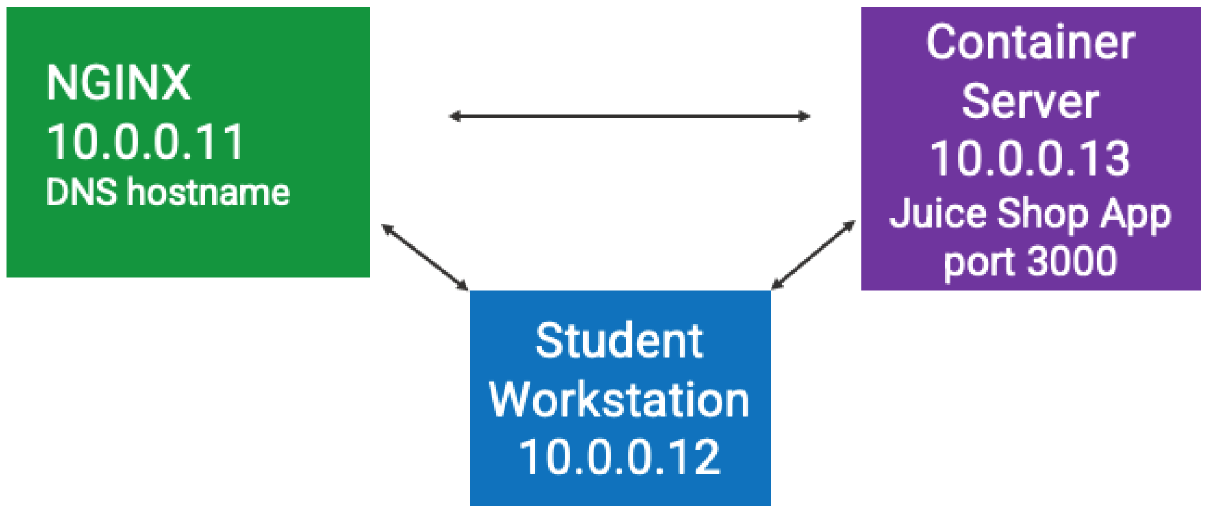
Labs - Secure API Traffic using JWT Authentication

Lab Contents

1. **Create auth_jwt realm and encode the secret**
2. **Issue, sign and test the JWT**

Estimated time to complete labs: **[XX] minutes**

Lab Network Topology- Possibly Delete Topic?



[Narrative Text]

Lab 1: Create auth_jwt Realm and Encode the Secret

Estimated time for completion: **XX minutes**

Requirements

The following tasks must be completed before beginning this lab:

- The configuration file /etc/nginx/conf.d/api_server.conf must exist.

Objectives

At the end of this lab you will be able to:

- Create an auth_jwt realm and encode a secret.
- Issue, sign and test the JWT

Steps

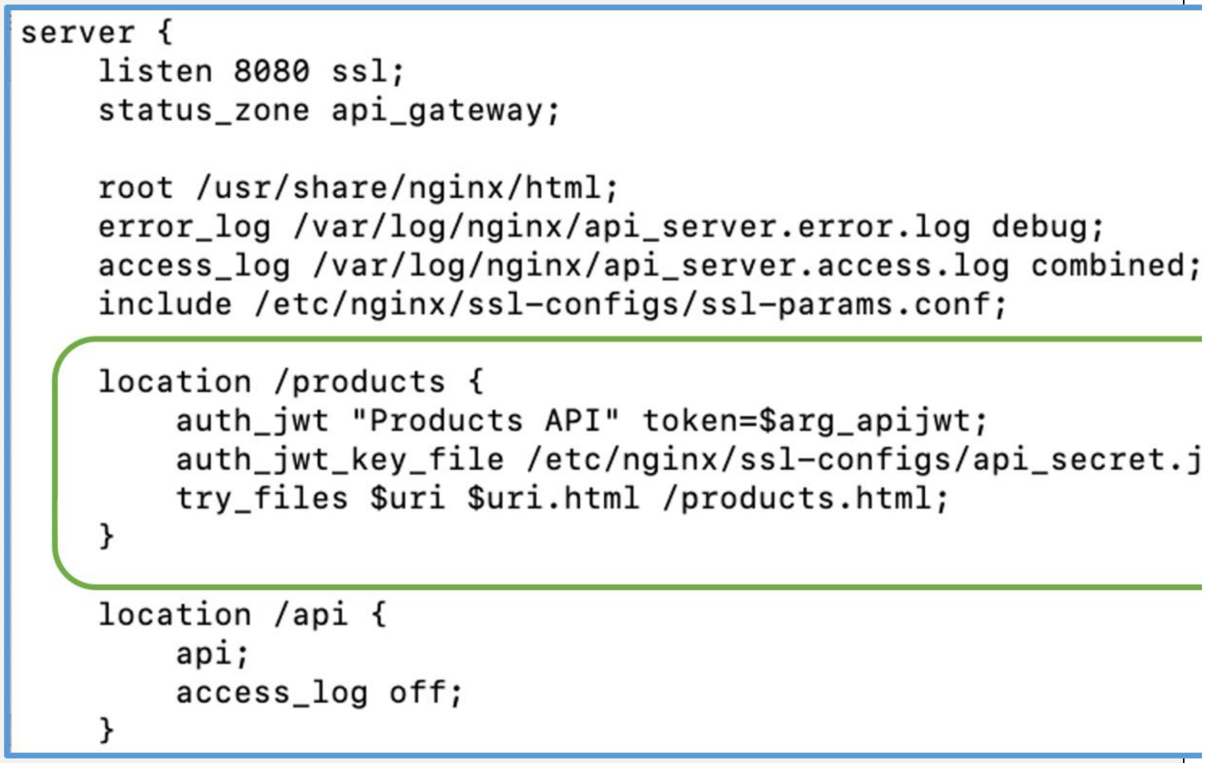
7. Edit the **api_server.conf** file.

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

8. In the 8080 server context, create the **/products** location with the following **auth_jwt** and **auth_jwt_key_file** values. Save the file.

```
location /products {
    auth_jwt "Products API" token=$arg_apijwt;
    auth_jwt_key_file /etc/nginx/ssl-configs/api_secret.jwk;
    try_files $uri $uri.html /products.html;
}
```

NOTE
The output should be:



```
server {
    listen 8080 ssl;
    status_zone api_gateway;

    root /usr/share/nginx/html;
    error_log /var/log/nginx/api_server.error.log debug;
    access_log /var/log/nginx/api_server.access.log combined;
    include /etc/nginx/ssl-configs/ssl-params.conf;

    location /products {
        auth_jwt "Products API" token=$arg_apijwt;
        auth_jwt_key_file /etc/nginx/ssl-configs/api_secret.j
        try_files $uri $uri.html /products.html;
    }

    location /api {
        api;
        access_log off;
    }
```

9. Use **base64** to create a JSON Web Key (symmetric key used for signing or a secret) using the secret “**fantasticjwt**”. The **tr** command makes the required character substitutions for Base64URL encoding. Copy the command from the **SNIPPETS** file.

```
$ echo -n fantasticjwt | base64 | tr '+/' '-_' | tr -d '='
```

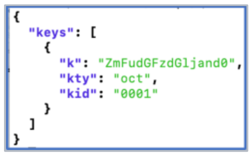
The output should be: `ZmFudGFzdGljand0`

10. Create the file **api_secret.jwk** with the JSON Web Key (secret) to be used for signing.

```
$ echo ‘{“keys”:[{“k”:”<result from encoded
secret>”,”kty”:”oct”,”kid”:”0001”}]}’ | sudo tee /etc/nginx/ssl-
configs/api_secret.jwk
```

NOTE
If you use “**fantasticjwt**” as your secret, you can just copy the command from the **SNIPPETS** file. Otherwise, you must replace the encoded secret with your value from the previous lab step.

Your symmetric key contains these 3 values:



```
{
  "keys": [
    {
      "k": "ZmFudGFzdGljand0",
      "kty": "oct",
      "kid": "0001"
    }
  ]
}
```

The “**k**” field contains a **base-64 value** of the <***encoded secret***>from the output of the previous step.
The “**kty**” field contains **oct** for the octet sequence.
The “**kid**” field contains **0001** for the serial number.

11. To view the information in a nicer format:

```
$ cat /etc/nginx/ssl-configs/api_secret.jwk | jq
```

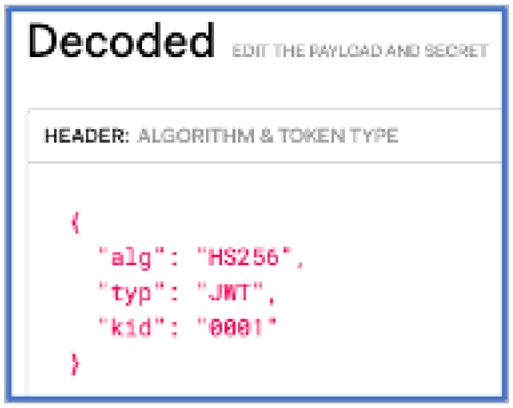
12. Test your NGINX configuration/syntax.

```
$ sudo nginx -t
```

13. Use **https://jwt.io** to create the JWT header, payload and signature. Scroll down to get to the ***Decoded*** section of the webpage.

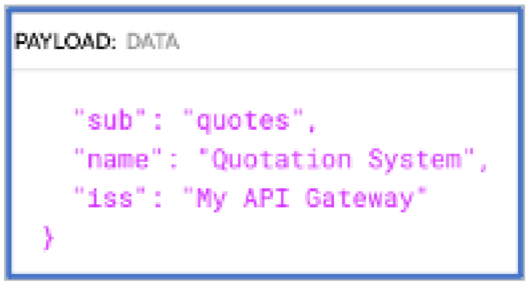
    https://jwt.io

14. To create the JWT **HEADER** copy the code from the SNIPPETS file and enter into the HEADER section on the right side of the webpage under ***Decoded*** replacing the existing header information.



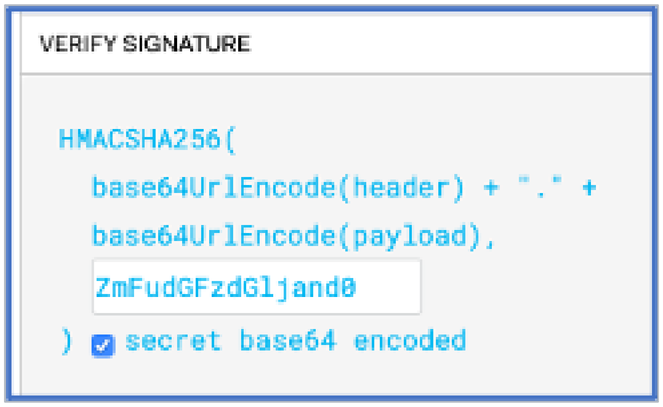
```
{
  "alg": "HS256",
  "typ": "JWT",
  "kid": "0001"
}
```

15. To create the JWT **PAYLOAD** copy the code from the SNIPPETS file and enter into the PAYLOAD section replacing the existing information.



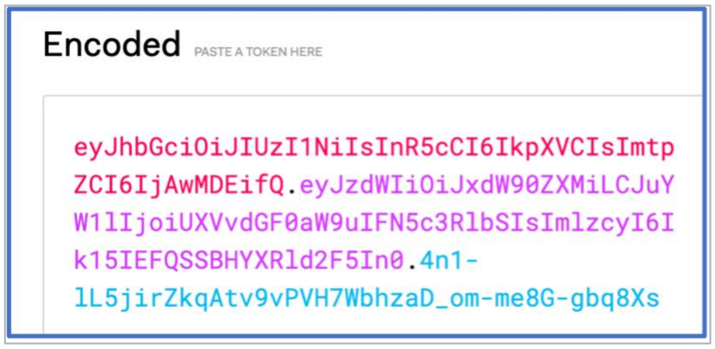
```
  "sub": "quotes",
  "name": "Quotation System",
  "iss": "My API Gateway"
}
```

16. To create the JWT **SIGNATURE** copy the code from the SNIPPETS file or from your previous lab step where you encoded the secret "fantasticjwt" and enter it into the *Verify SIGNATURE* section. Make sure to select **secret base64 encoded**.



```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  ZmFudGFzdGljand0
) ✓ secret base64 encoded
```

17. On the left side in the **Encoded** section you should have your base64 encoded **HEADER.PAYLOAD.SIGNATURE.** Copy this to use in testing access to your system.

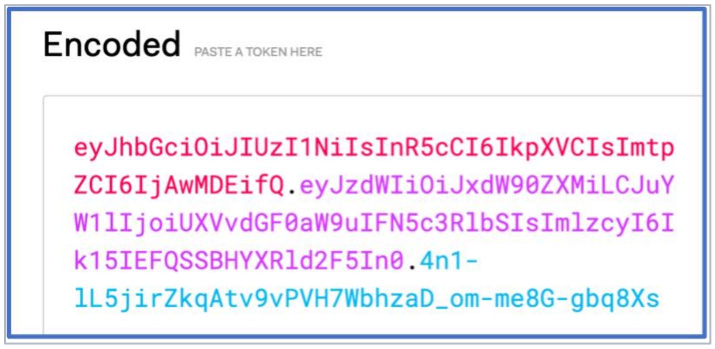

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtp
ZCI6IjAwMDEifQ.eyJzdWIiOiJxdW90ZXMiLCJuY
W1lIjoiUXVvdGF0aW9uIFN5c3RlbSIsImlzcyI6I
k15IEFQSSBHYXRld2F5In0.4n1-
lL5jirZkqAtv9vPVH7WbhzaD_om-me8G-gbq8Xs
```

NOTE
Below the Encoded box, the signature is now verified.

18. Create the file **quotes.jwt** with the encoded **HEADER.PAYLOAD.SIGNATURE** that you created. You can copy from the SNIPPETS file or from your jwt.io browser.

```
$ sudo vim /etc/nginx/ssl-configs/quotes.jwt
```


Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtp
ZCI6IjAwMDEifQ.eyJzdWIiOiJxdW90ZXMiLCJuY
W1lIjoiUXVvdGF0aW9uIFN5c3RlbSIsImlzcyI6I
k15IEFQSSBHYXRld2F5In0.4n1-
lL5jirZkqAtv9vPVH7WbhzaD_om-me8G-gbq8Xs
```

NOTE
The HEADER is in red, the PAYLOAD in magenta and the SIGNATURE in turquoise.

19. Create the file **products.html** in /usr/share/nginx/html.

```
$ sudo echo "<h1>JWT Token WORKS</h1><p>Your JWT Authentication is
successful</p>" | sudo tee /usr/share/nginx/html/products.html
```

20. Reload NGINX.

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

21. Use the curl command to request /products.

```
$ curl -k https://www.nginxtraining.com:8080/products
```

NOTE
You should receive a **401 Authorization Required** response because you set /products to. now require a JWT.

22. Test by appending the file quotes.jwt to the URI argument apijwt.

```
$ curl -k https://www.nginxtraining.com:8080/products?apijwt=`cat
/etc/nginx/ssl-configs/quotes.jwt`
```

NOTE
The output should be:
<h1>JWT Token Works</h1><p>Your JWT Authentication is successful</p>

Expected Results

NGINX has been configured to use a JWT for authentication in order to access a particular URI.