PKI Assignment

Configuration of HTTPS in NGINX

By

Aravind Kumar (MSc. Computer Security)

Description

TLS (Transport Layer Security) and SSL (Secure Socket Layer) are web protocols used to encrypt the traffic between server and client without the possibility of the message being intercepted by third person. The certificate system also assists the users in verifying the identity of the sites that are connected with.

We are going to see how to set up a self-signed SSL certificate for use with a NGINX web server.

So, what is NGINX?

- NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server.
- It is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption.
- Unlike traditional servers, NGINX doesn't rely on threads to handle requests. Instead it uses a much more scalable event-driven (asynchronous) architecture.
- This architecture uses small, but more importantly, predictable amounts of memory under load. Even if you don't expect to handle thousands of simultaneous requests, you can still benefit from NGINX's high-performance and small memory footprint.
- NGINX scales in all directions: from the smallest VPS all the way up to large clusters of servers.

First we are going to see the installation of NGINX and then we are going to configure the HTTPS in NGINX for the secure connection of the traffic.

NGINX Installation

The following are the steps to install the NGINX in the Ubuntu machine.

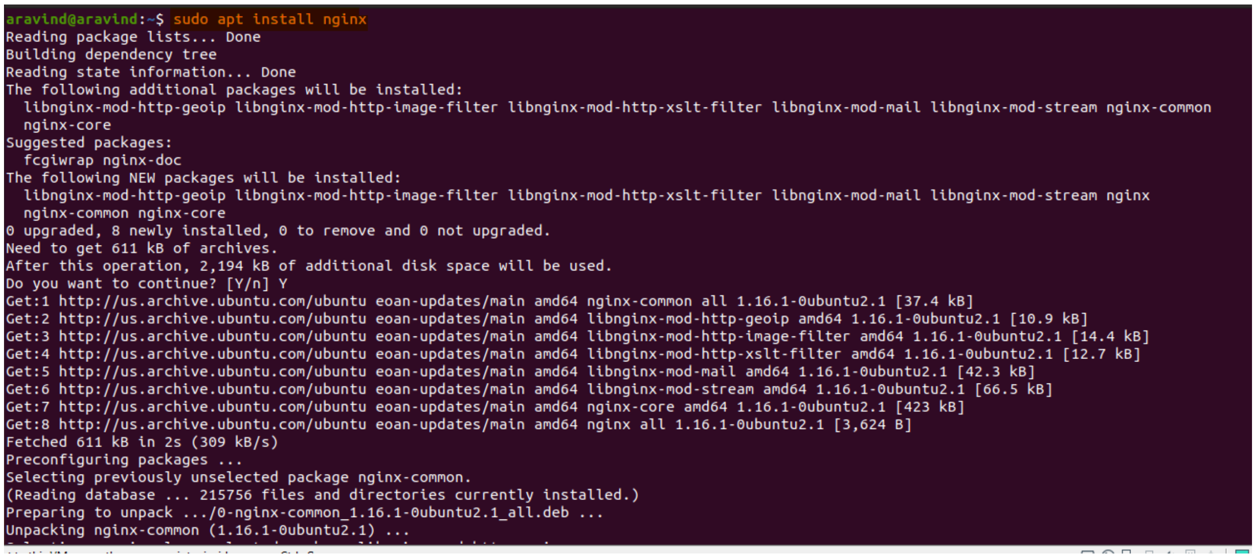
Steps 1 – Prerequisites

- We must have a non-root user configured with sudo privileges configured on our server.

Step 2 – Installing NGINX

- Since NGINX is available by default in Ubuntu repositories, it is possible to install it from the repositories using the apt packaging system.

  Command – sudo apt install nginx



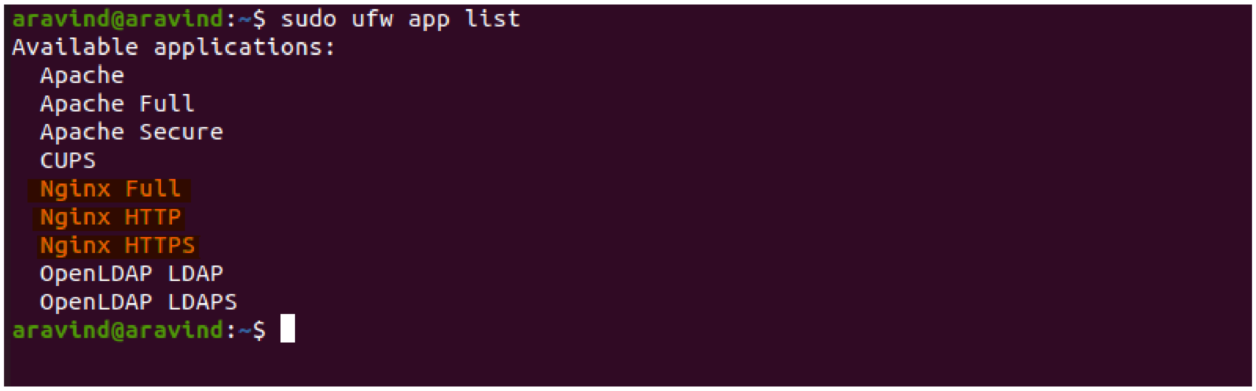
```
aravind@aravind:~$ sudo apt install nginx
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libnginx-mod-http-geoip libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream nginx-common
  nginx-core
Suggested packages:
  fcgiwrap nginx-doc
The following NEW packages will be installed:
  libnginx-mod-http-geoip libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream nginx
  nginx-common nginx-core
0 upgraded, 8 newly installed, 0 to remove and 0 not upgraded.
Need to get 611 kB of archives.
After this operation, 2,194 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 nginx-common all 1.16.1-0ubuntu2.1 [37.4 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 libnginx-mod-http-geoip amd64 1.16.1-0ubuntu2.1 [10.9 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 libnginx-mod-http-image-filter amd64 1.16.1-0ubuntu2.1 [14.4 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 libnginx-mod-http-xslt-filter amd64 1.16.1-0ubuntu2.1 [12.7 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 libnginx-mod-mail amd64 1.16.1-0ubuntu2.1 [42.3 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 libnginx-mod-stream amd64 1.16.1-0ubuntu2.1 [66.5 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 nginx-core amd64 1.16.1-0ubuntu2.1 [423 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu eoan-updates/main amd64 nginx all 1.16.1-0ubuntu2.1 [3,624 B]
Fetched 611 kB in 2s (309 kB/s)
Preconfiguring packages ...
Selecting previously unselected package nginx-common.
(Reading database ... 215756 files and directories currently installed.)
Preparing to unpack .../0-nginx-common_1.16.1-0ubuntu2.1_all.deb ...
Unpacking nginx-common (1.16.1-0ubuntu2.1) ...
```

Step 3 – Adjusting the Firewall

- Before using NGINX, we must make some changes in the firewall to allow access to the services.
- NGINX registers itself as a service with ***ufw*** upon installation, making it straight forward to allow access

  Command – sudo ufw app list

  When we enter the command, the following should appear

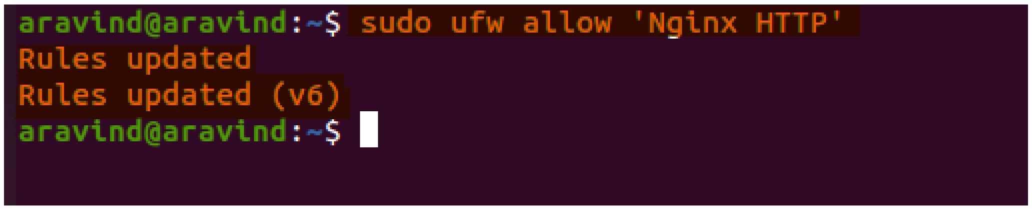


```
aravind@aravind:~$ sudo ufw app list
Available applications:
  Apache
  Apache Full
  Apache Secure
  CUPS
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenLDAP LDAP
  OpenLDAP LDAPS
aravind@aravind:~$
```

- As you can see, there are three profiles available for Nginx:
  - ➢ **Nginx Full**: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)
  - ➢ **Nginx HTTP**: This profile opens only port 80 (normal, unencrypted web traffic)
  - ➢ **Nginx HTTPS**: This profile opens only port 443 (TLS/SSL encrypted traffic)

- Since we haven't configured SSL for our server, we will only need to allow traffic on port 80.

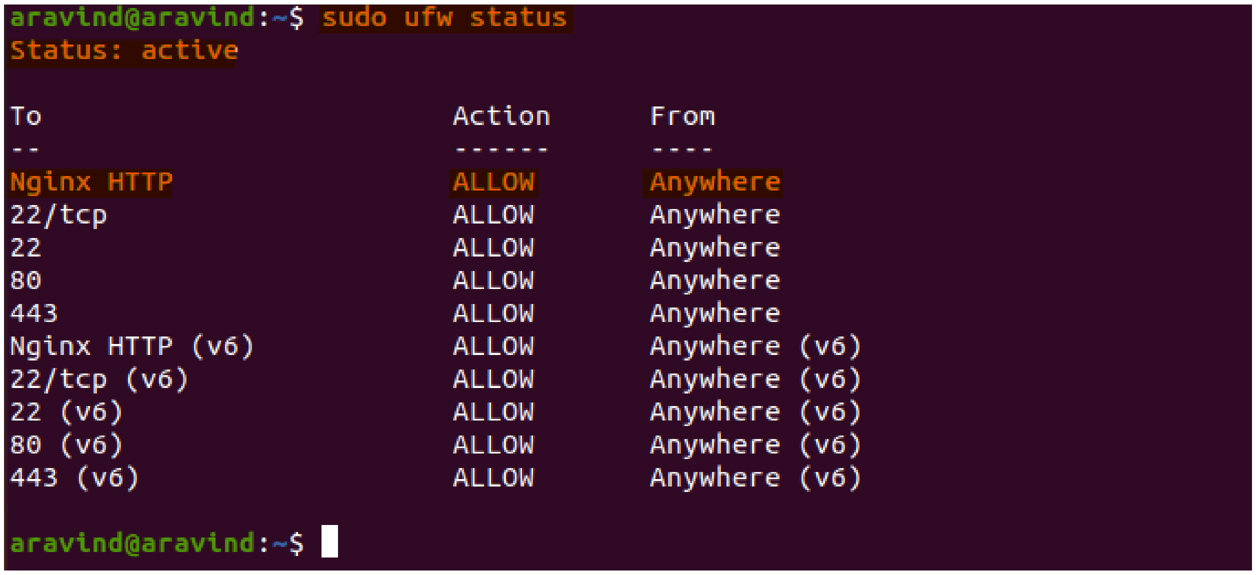
  Command – sudo ufw allow 'Nginx HTTP'



```
aravind@aravind:~$ sudo ufw allow 'Nginx HTTP'
Rules updated
Rules updated (v6)
aravind@aravind:~$
```

- To verify it the changes, type the following command

  Command – sudo ufw status

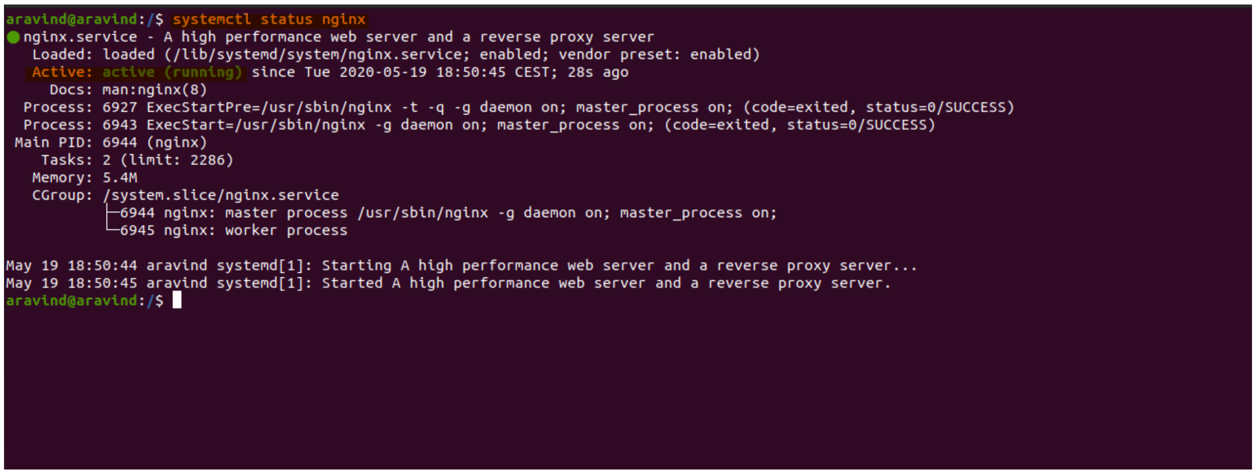


```
aravind@aravind:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Nginx HTTP                 ALLOW       Anywhere
22/tcp                     ALLOW       Anywhere
22                         ALLOW       Anywhere
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
22/tcp (v6)                ALLOW       Anywhere (v6)
22 (v6)                    ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)

aravind@aravind:~$
```

Step 4 – Checking your Web Server

- After installing NGINX, the web server should already be up and running. We can check by using the following command

  Command – systemctl status nginx



```
aravind@aravind:/$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-05-19 18:50:45 CEST; 28s ago
     Docs: man:nginx(8)
  Process: 6927 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
  Process: 6943 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
 Main PID: 6944 (nginx)
    Tasks: 2 (limit: 2286)
   Memory: 5.4M
   CGroup: /system.slice/nginx.service
           ├─6944 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
           └─6945 nginx: worker process

May 19 18:50:44 aravind systemd[1]: Starting A high performance web server and a reverse proxy server...
May 19 18:50:45 aravind systemd[1]: Started A high performance web server and a reverse proxy server.
aravind@aravind:/$
```

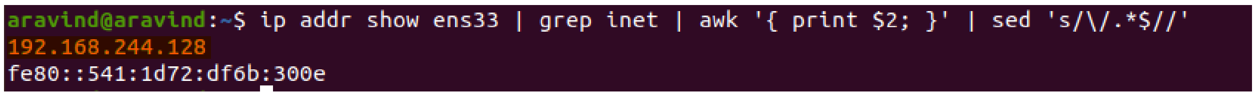
Note

If you have apache service running in your machine stop it so that there will not be any error while starting the application.

- As we can able to see that service appears to have started successfully but the best way to test this is to actually request a page from NGINX.

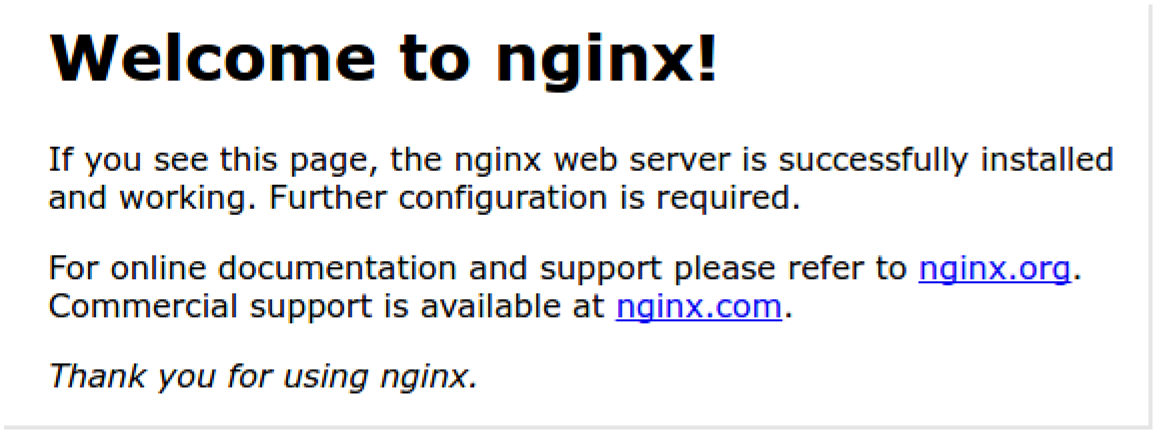
  Command – ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'

  This will show the ip address of the server.



```
aravind@aravind:~$ ip addr show ens33 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'
192.168.244.128
fe80::541:1d72:df6b:300e
```

- Enter the IP address in the browser and you will have the following message



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

Conclusion

We have installed the NGINX in the ubuntu machine and also have configured the firewall.

Configuration of HTTPS in NGINX

Step 1 – Prerequisite

- We must have a non-root user configured with sudo privileges configured on our server.

Step 2 – Creating the SSL Certificate

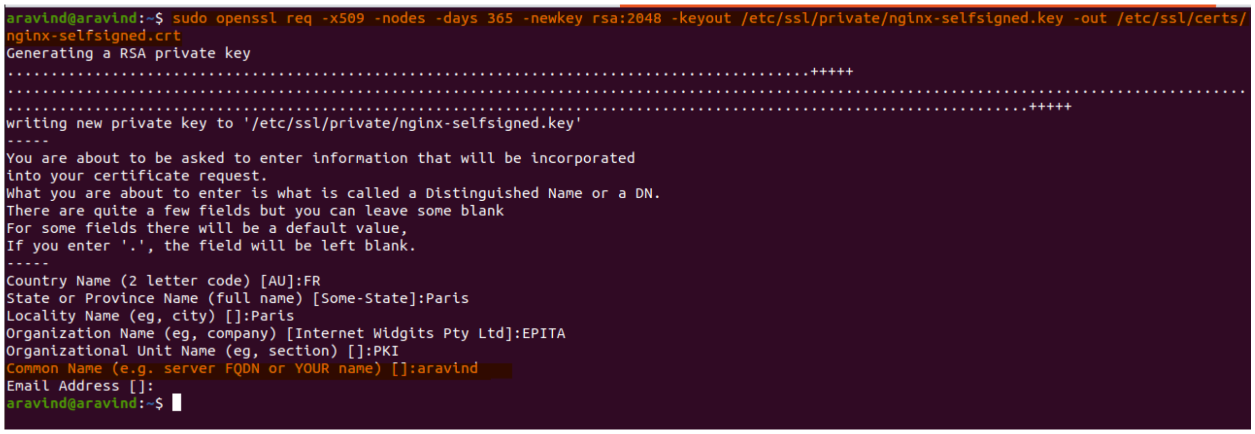
- SSL works by using a combination of a public certificate and a private key.
- The SSL key is kept secret on the server. It is used to encrypt content sent to clients.
- The public SSL certificate is publicly shared with anyone requesting the content.
- It can be used to decrypt the content signed by the associated SSL key.
- We can create a self-signed key and certificate pair with OpenSSL in a single command
  Command - sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt

Synopsis

- **openssl**: This is the basic command line tool for creating and managing OpenSSL certificates, keys, and other files.
- **req**: This subcommand specifies that we want to use X.509 certificate signing request (CSR) management. The "X.509" is a public key infrastructure standard that SSL and TLS adhere to for its key and certificate management. We want to create a new X.509 cert, so we are using this subcommand.
- **-x509**: This further modifies the previous subcommand by telling the utility that we want to make a self-signed certificate instead of generating a certificate signing request, as would normally happen.
- **-nodes**: This tells OpenSSL to skip the option to secure our certificate with a passphrase. We need Nginx to be able to read the file, without user intervention, when the server starts up. A passphrase would prevent this from happening because we would have to enter it after every restart.
- **-days 365**: This option sets the length of time that the certificate will be considered valid. We set it for one year here.
- **-newkey rsa:2048**: This specifies that we want to generate a new certificate and a new key at the same time. We did not create the key that is required to sign the

certificate in a previous step, so we need to create it along with the certificate. The rsa:2048 portion tells it to make an RSA key that is 2048 bits long.

- **-keyout**: This line tells OpenSSL where to place the generated private key file that we are creating.
- **-out**: This tells OpenSSL where to place the certificate that we are creating.

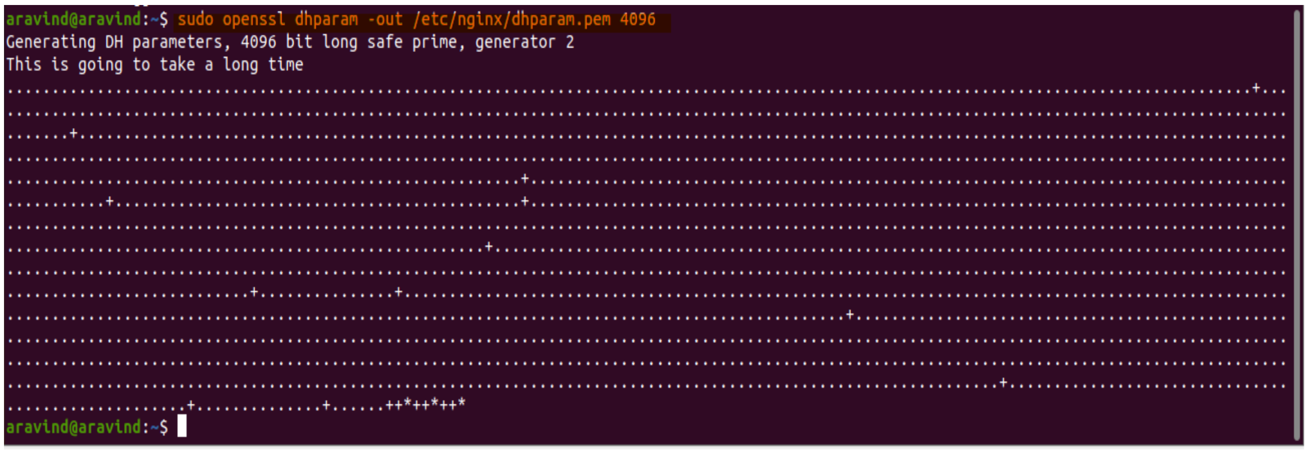


```
aravind@aravind:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/
nginx-selfsigned.crt
Generating a RSA private key
.........................................................................................+++++
................................................................................................................................................
...............................................................................................................................+++++
writing new private key to '/etc/ssl/private/nginx-selfsigned.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Paris
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:EPITA
Organizational Unit Name (eg, section) []:PKI
Common Name (e.g. server FQDN or YOUR name) []:aravind
Email Address []:
aravind@aravind:~$
```

- Fill out the prompts appropriately. The most important one is that request the common name (e.g. server FEDN or YOUR name)
- Both the files we created will be placed in the appropriate subdirectories of the **/etc/ssl** directory.
- While we are using OpenSSL, we should also create a strong Diffie-Hellman group, which is used in negotiating Perfect Forward Secrecy with the clients. (Refer the link to know more about Perfect Forward Secrecy).

  Command - sudo openssl dhparam -out /etc/nginx/dhparam.pem 4096

  This will take some time, but when it's done, we will have a strong DH group at **/etc/nginx/dhparam.pem** that we can use in our configuration.



```
aravind@aravind:~$ sudo openssl dhparam -out /etc/nginx/dhparam.pem 4096
Generating DH parameters, 4096 bit long safe prime, generator 2
This is going to take a long time
....................................................................................................................+...
.........................................................................................................................
......+..................................................................................................................
.........................................................................................................................
.................................................+.......................................................................
..........+.......................................+......................................................................
.........................................................................................................................
.............................................+...........................................................................
.........................................................................................................................
.......................+.............+...................................................................................
.......................................................................................+.................................
.........................................................................................................................
.........................................................................................................................
...................................................................................................+.....................
.................+............+......++*++*++*
aravind@aravind:~$
```

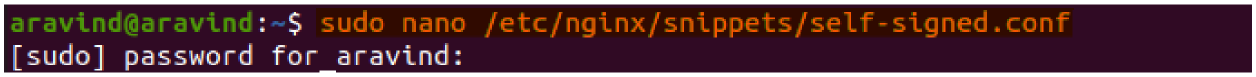
Step 3 – Configuring NGINX to use SSL

- Since we have created our key and certificate under the **/etc/ssl** directory all we need to do now is to modify our NGINX configuration to take advantage of it.
- Adjustments to our configurations:
    A. We will create a configuration snippet containing our SSL key and certificate file locations.
    B. We will create a configuration snippet containing strong SSL settings that can be used with any certificate in the future.
    C. We will adjust our NGINX server blocks to handle SSL requests and use the two snippets above.
- By doing the above configuration the NGINX will allows us to keep clean server blocks and put common configuration segments into reusable modules.

A. Creating a configuration snippet pointing to the SSL key and certificate

- First, we will create a new NGINX configuration snippet in the **/etc/nginx/**snippets directory and to distinguish this file, let's name it **self-signed.conf**

  Command - sudo nano /etc/nginx/snippets/self-signed.conf



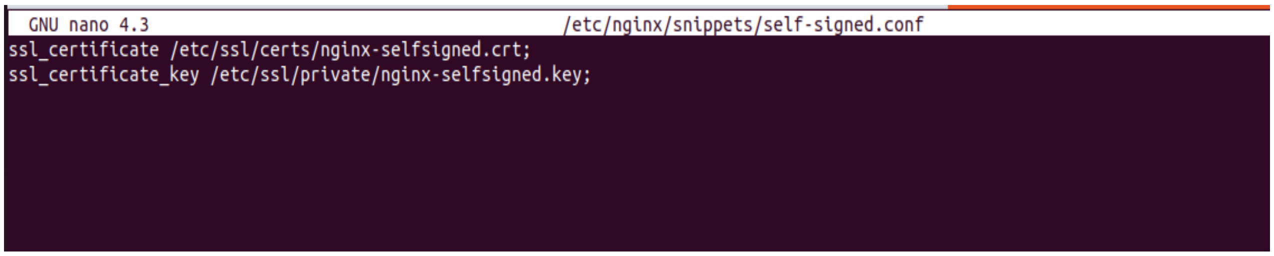
```
aravind@aravind:~$ sudo nano /etc/nginx/snippets/self-signed.conf
[sudo] password for aravind:
```

Inside the file we need to set the **ssl_certificate** directive to our certificate file and the **ssl_certificate_key** to the associated key.

Enter the below lines in the file

ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;



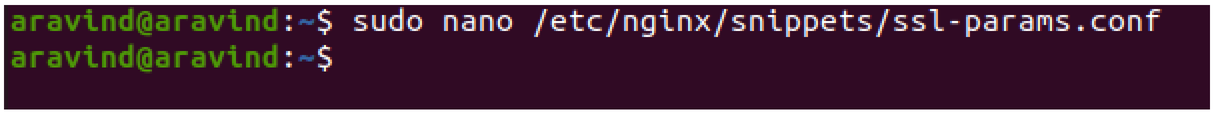
```
  GNU nano 4.3                    /etc/nginx/snippets/self-signed.conf
ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;
```

After adding the lines, save and close the file.

B. Creating a configuration snippet with strong encryption settings

- Now we will create another snippet that will define SSL settings. This will set NGINX up with a strong SSL cipher suite and enable some advanced features that will help keep our server secure.
- The parameter we will set can be reused in future NGINX configuration, so we will give the file a generic name

Command - sudo nano /etc/nginx/snippets/ssl-params.conf



```
aravind@aravind:~$ sudo nano /etc/nginx/snippets/ssl-params.conf
aravind@aravind:~$
```

Copy the following in the **ssl-params.conf**

ssl_protocols TLSv1.2;

ssl_prefer_server_ciphers on;

ssl_dhparam /etc/nginx/dhparam.pem;

ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384;

ssl_ecdh_curve secp384r1; # Requires nginx >= 1.1.0

ssl_session_timeout 10m;

ssl_session_cache shared:SSL:10m;

ssl_session_tickets off; # Requires nginx >= 1.5.9

ssl_stapling on; # Requires nginx >= 1.3.7

ssl_stapling_verify on; # Requires nginx => 1.3.7

resolver 8.8.8.8 8.8.4.4 valid=300s;

resolver_timeout 5s;

# Disable strict transport security for now. You can uncomment the following

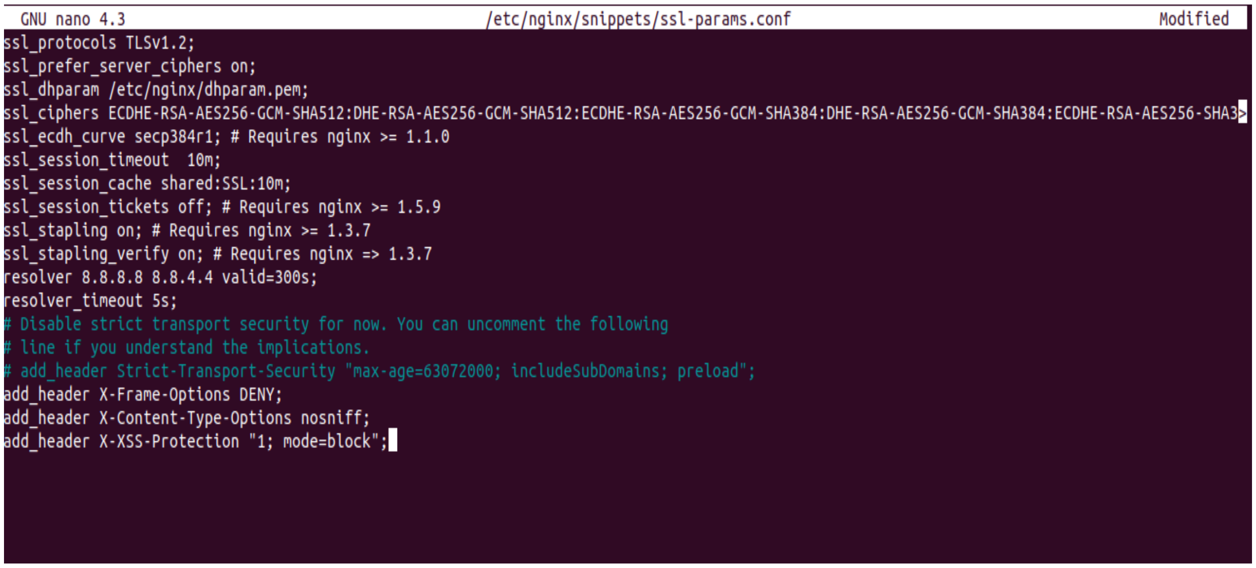
# line if you understand the implications.

# add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload";

add_header X-Frame-Options DENY;

add_header X-Content-Type-Options nosniff;

add_header X-XSS-Protection "1; mode=block";



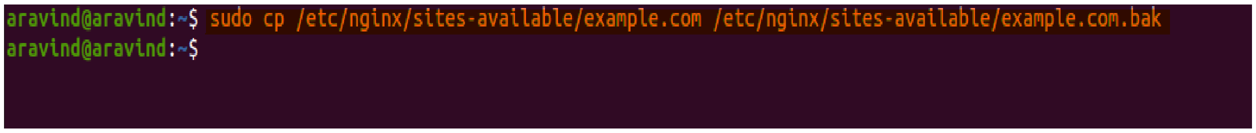
```
  GNU nano 4.3                         /etc/nginx/snippets/ssl-params.conf                          Modified
ssl_protocols TLSv1.2;
ssl_prefer_server_ciphers on;
ssl_dhparam /etc/nginx/dhparam.pem;
ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA3>
ssl_ecdh_curve secp384r1; # Requires nginx >= 1.1.0
ssl_session_timeout  10m;
ssl_session_cache shared:SSL:10m;
ssl_session_tickets off; # Requires nginx >= 1.5.9
ssl_stapling on; # Requires nginx >= 1.3.7
ssl_stapling_verify on; # Requires nginx => 1.3.7
resolver 8.8.8.8 8.8.4.4 valid=300s;
resolver_timeout 5s;
# Disable strict transport security for now. You can uncomment the following
# line if you understand the implications.
# add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload";
add_header X-Frame-Options DENY;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";
```

- Because we are using self-signed certificate, the SSL stapling will not be used.

C. Adjusting the NGINX configuration to use SSL

- Now that we have our snippets, we can adjust our NGINX configuration to enable SSL.
- We will assume in this guide that you are using a custom server block configuration file in the **/etc/nginx/**sites-available directory. We will use **/etc/nginx/sites-available/example.com** for this example. Substitute your configuration filename as needed.
- Let us backup our current configuration file before we proceed further

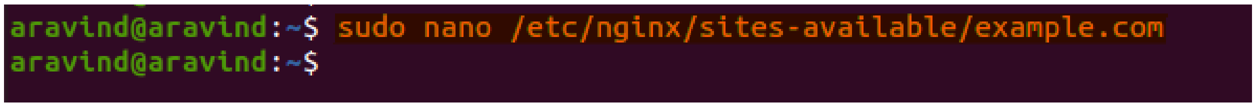
  Command - sudo cp /etc/nginx/sites-available/example.com /etc/nginx/sites-available/example.com.bak



```
aravind@aravind:~$ sudo cp /etc/nginx/sites-available/example.com /etc/nginx/sites-available/example.com.bak
aravind@aravind:~$
```

- Now, open the configuration file to make changes

  Command - sudo nano /etc/nginx/sites-available/example.com



```
aravind@aravind:~$ sudo nano /etc/nginx/sites-available/example.com
aravind@aravind:~$
```

Now replace the lines which is inside the file by the below lines

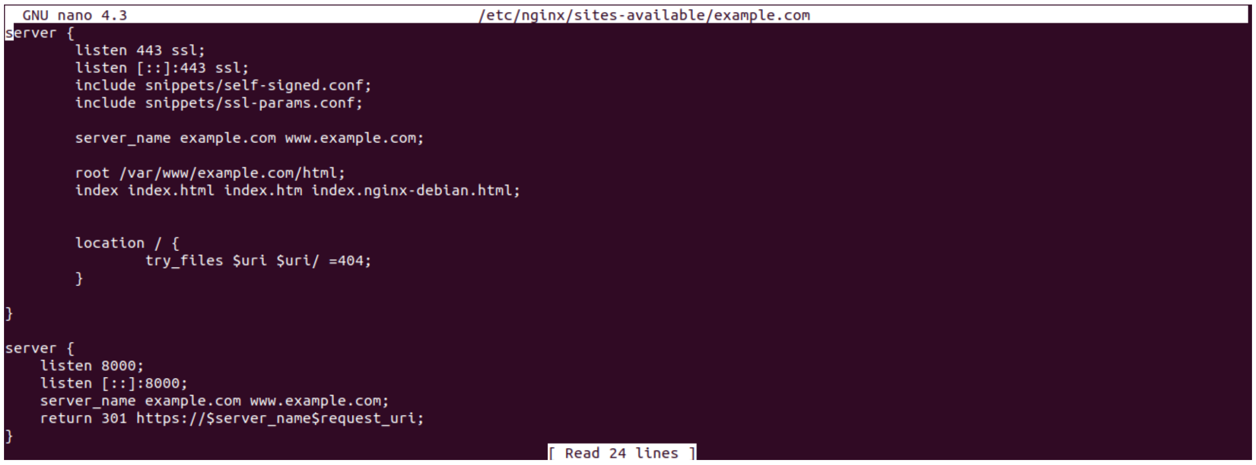
```
server {
    listen 443 ssl;
    listen [::]:443 ssl;
    include snippets/self-signed.conf;
    include snippets/ssl-params.conf;
    root /var/www/example.com/html;
    index index.html index.htm index.nginx-debian.html;
    server_name example.com www.example.com;
    location / {
        try_files $uri $uri/ =404;
    }
}
server {
  listen 8000;
```

listen [::]:8000;

server_name example.com www.example.com;

return 301 https://$server_name$request_uri;

}

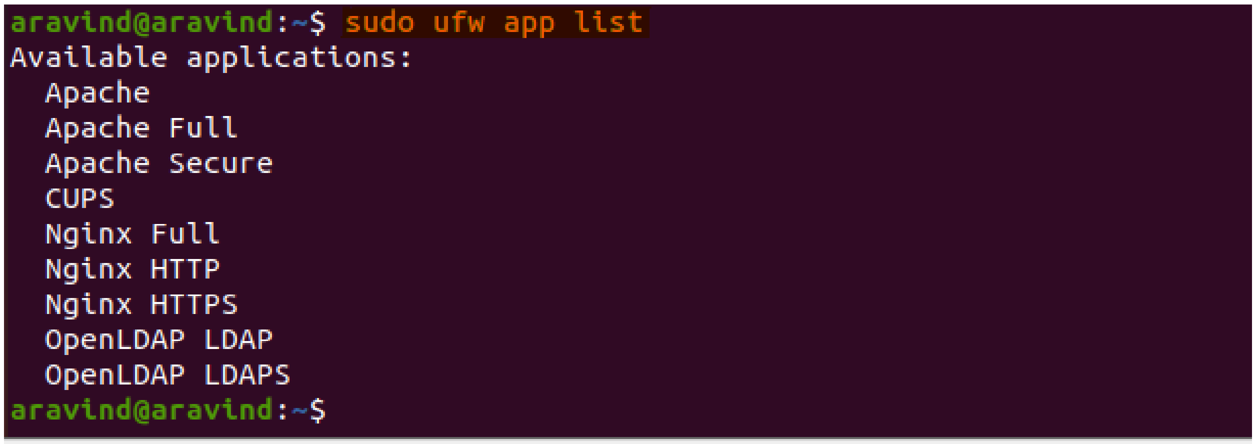


```
  GNU nano 4.3                                          /etc/nginx/sites-available/example.com
server {
        listen 443 ssl;
        listen [::]:443 ssl;
        include snippets/self-signed.conf;
        include snippets/ssl-params.conf;

        server_name example.com www.example.com;

        root /var/www/example.com/html;
        index index.html index.htm index.nginx-debian.html;


        location / {
                try_files $uri $uri/ =404;
        }

}

server {
    listen 8000;
    listen [::]:8000;
    server_name example.com www.example.com;
    return 301 https://$server_name$request_uri;
}
                                                              [ Read 24 lines ]
```

Step 4 – Adjusting the Firewall

- Now we need to adjust the settings to allow for SSL traffic. Luckily, NGINX registers a few pofiles with **ufw** upon installation.

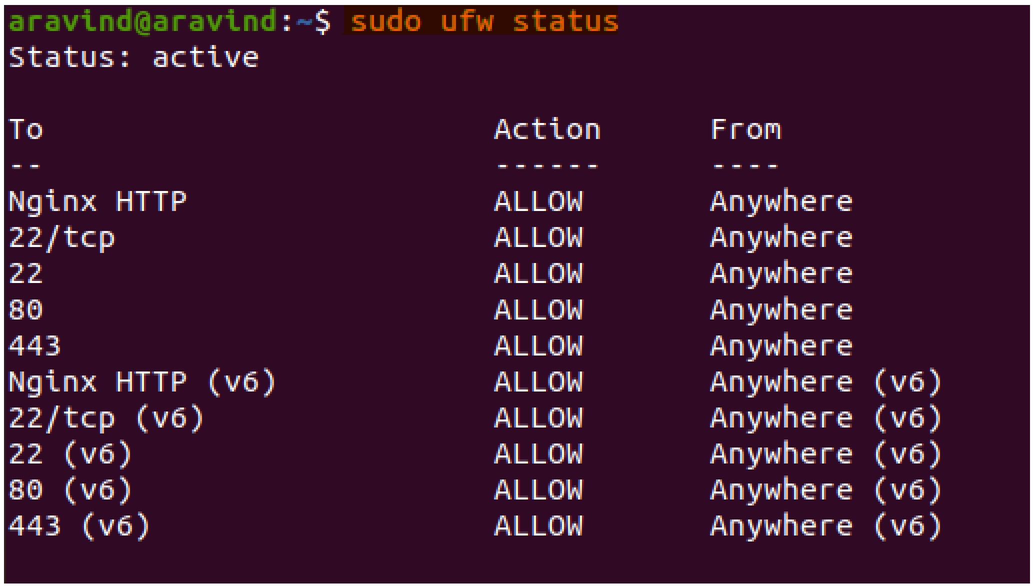
  Command – sudo ufw app list



```
aravind@aravind:~$ sudo ufw app list
Available applications:
  Apache
  Apache Full
  Apache Secure
  CUPS
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenLDAP LDAP
  OpenLDAP LDAPS
aravind@aravind:~$
```

- To see the current settings, type the below command

  Command – sudo ufw status



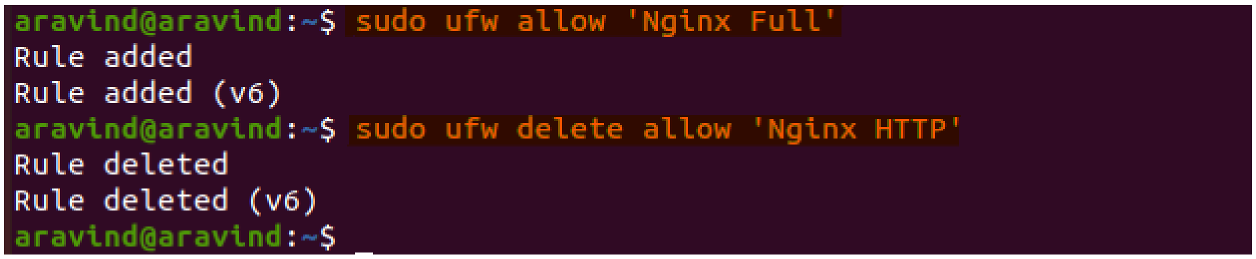
```
aravind@aravind:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Nginx HTTP                 ALLOW       Anywhere
22/tcp                     ALLOW       Anywhere
22                         ALLOW       Anywhere
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
22/tcp (v6)                ALLOW       Anywhere (v6)
22 (v6)                    ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
```

- To additionally let the HTTPS traffic, we can allow the **"Nginx Full"** profile and then delete the redundant **"Nginx HTTP"** profile allowance by the below commands

  Command

  sudo ufw allow 'Nginx Full'

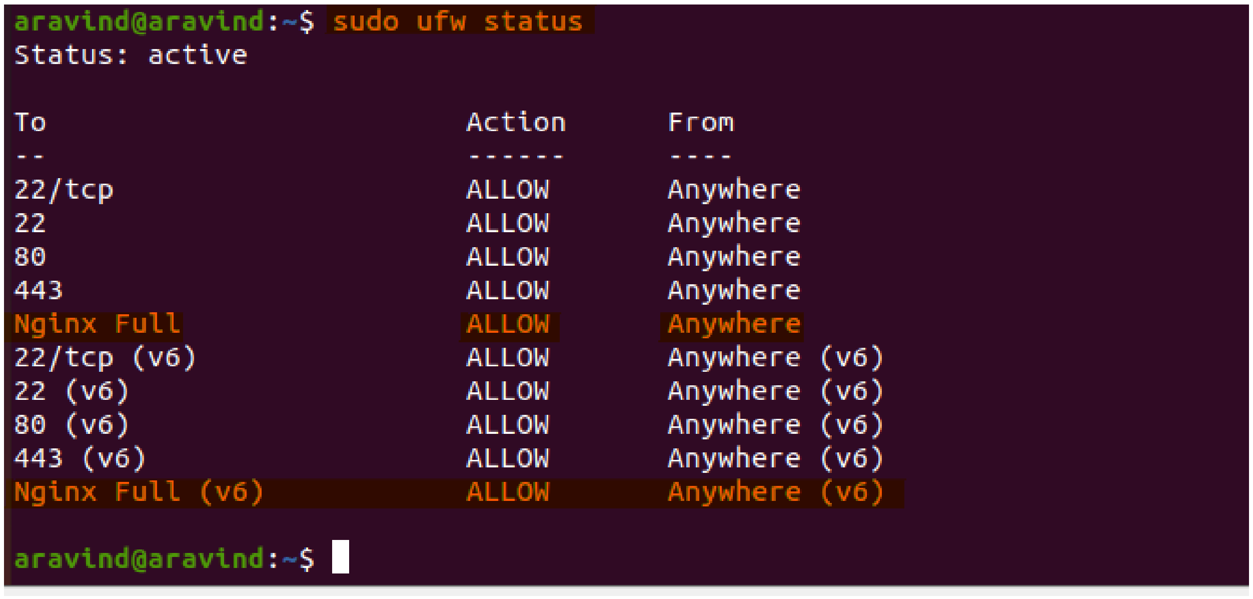
  sudo ufw delete allow 'Nginx HTTP'



```
aravind@aravind:~$ sudo ufw allow 'Nginx Full'
Rule added
Rule added (v6)
aravind@aravind:~$ sudo ufw delete allow 'Nginx HTTP'
Rule deleted
Rule deleted (v6)
aravind@aravind:~$ _
```

- Now the status will look like

  Command – sudo ufw status

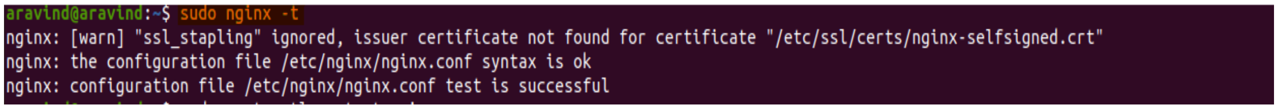


```
aravind@aravind:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
22                         ALLOW       Anywhere
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
Nginx Full                 ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
22 (v6)                    ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
Nginx Full (v6)            ALLOW       Anywhere (v6)

aravind@aravind:~$
```

Step 5 – Enabling the changes in NGINX

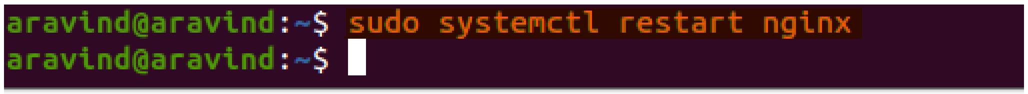
- Now that we've made our changes and adjusted our firewall, we can restart NGINX to implement our changes.
- Before all that we need to check whether we don't have any syntax errors in our files. To check that type the following command

  Command – sudo nginx -t



```
aravind@aravind:~$ sudo nginx -t
nginx: [warn] "ssl_stapling" ignored, issuer certificate not found for certificate "/etc/ssl/certs/nginx-selfsigned.crt"
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

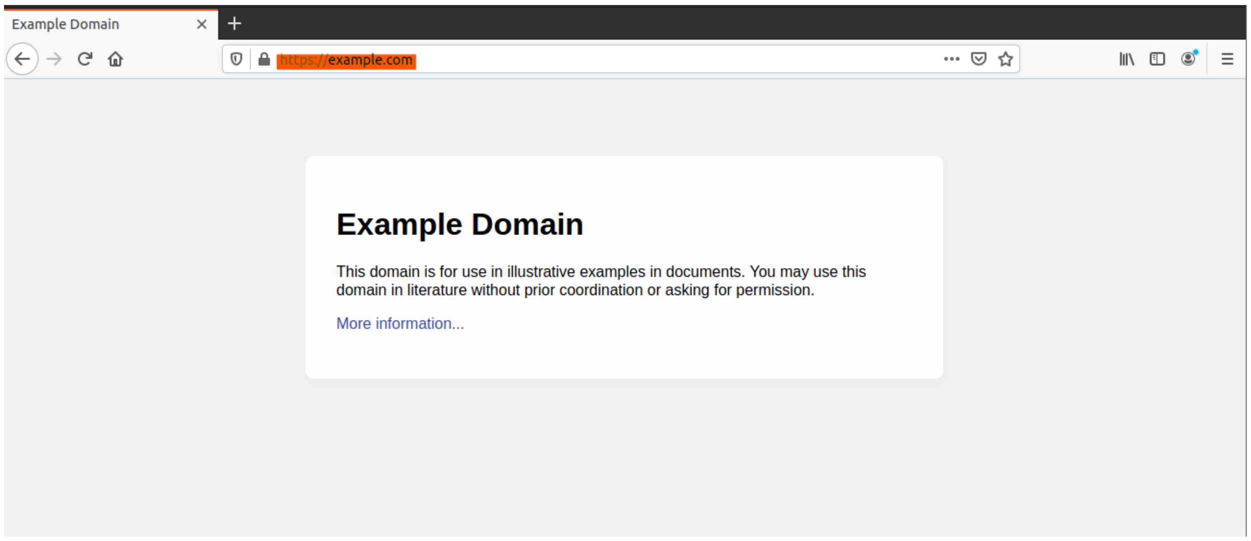
- If success, the result will be like the above output
- The warning is due to the self-signed certificate because it can't use SSL stapling. This is expected and our sever can still encrypt connections correctly.
- Now restart the NGINX to implement the changes
  Command – sudo systemctl restart nginx



```
aravind@aravind:~$ sudo systemctl restart nginx
aravind@aravind:~$
```

Step 6 – Testing the Encryption

- Now we are going to test the SSL server.
- Open the browser and type the IP address of the server domain.
- You will get the below screen



Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...

- When entered the IP address it automatically changes to the redirection site which we have specified in the file **/etc/nginx/sites-available/example.com.**

Conclusion

Thus, we have configured the NGINX server to use the strong encryption for client connections. This will allow the server request securely and will prevent outside parties from reading the traffic.