1. Install Nginx and host a simple index.html with the message “Hello Nginx”
    - Let’s update our machine first using the following command:
        - *sudo apt-get update*

```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo apt-get update
[sudo] password for rikeshkarma:
Hit:1 https://download.docker.com/linux/ubuntu hirsute InRelease
Hit:2 http://ppa.launchpad.net/apandada1/brightness-controller/ubuntu hirsute InRelease
Hit:3 http://apt.pop-os.org/proprietary hirsute InRelease
Get:4 http://packages.microsoft.com/repos/code stable InRelease [10.4 kB]
Hit:5 http://dl.google.com/linux/chrome/deb stable InRelease
Hit:6 https://deb.nodesource.com/node_14.x hirsute InRelease
Hit:7 http://ppa.launchpad.net/papirus/papirus/ubuntu hirsute InRelease
Get:8 http://packages.microsoft.com/repos/code stable/main armhf Packages [55.7 kB]
Hit:9 http://us.archive.ubuntu.com/ubuntu hirsute InRelease
Get:10 http://packages.microsoft.com/repos/code stable/main arm64 Packages [55.9 kB]
Get:11 http://packages.microsoft.com/repos/code stable/main amd64 Packages [55.0 kB]
Hit:12 http://ppa.launchpad.net/system76/pop/ubuntu hirsute InRelease
Get:13 http://us.archive.ubuntu.com/ubuntu hirsute-security InRelease [110 kB]
Hit:14 https://brave-browser-apt-release.s3.brave.com stable InRelease
Get:15 http://us.archive.ubuntu.com/ubuntu hirsute-updates InRelease [115 kB]
Get:16 http://us.archive.ubuntu.com/ubuntu hirsute-proposed InRelease [269 kB]
Get:17 http://us.archive.ubuntu.com/ubuntu hirsute-backports InRelease [101 kB]
Get:18 http://us.archive.ubuntu.com/ubuntu hirsute-security/main amd64 DEP-11 Metadata [9,680 B]
Get:19 http://us.archive.ubuntu.com/ubuntu hirsute-security/universe amd64 DEP-11 Metadata [5,676 B]
Get:20 http://us.archive.ubuntu.com/ubuntu hirsute-updates/main amd64 DEP-11 Metadata [97.6 kB]
Get:21 http://us.archive.ubuntu.com/ubuntu hirsute-updates/universe amd64 DEP-11 Metadata [57.8 kB]
Get:22 http://us.archive.ubuntu.com/ubuntu hirsute-updates/multiverse amd64 DEP-11 Metadata [944 B]
Get:23 http://us.archive.ubuntu.com/ubuntu hirsute-proposed/main amd64 DEP-11 Metadata [10.4 kB]
Get:24 http://us.archive.ubuntu.com/ubuntu hirsute-proposed/universe amd64 DEP-11 Metadata [1,444 B]
Get:25 http://us.archive.ubuntu.com/ubuntu hirsute-backports/universe amd64 DEP-11 Metadata [9,348 B]
Fetched 965 kB in 4s (222 kB/s)
Reading package lists... Done
```

    - And to install ***Nginx*** we just need to invoke the following command in the terminal:
        - *sudo apt-get install nginx*

```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo apt-get install nginx
[sudo] password for rikeshkarma:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  linux-headers-5.11.0-7633
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libnginx-mod-http-geoip2 libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail
  libnginx-mod-stream libnginx-mod-stream-geoip2 nginx-common nginx-core
Suggested packages:
  fcgiwrap nginx-doc
The following NEW packages will be installed:
  libnginx-mod-http-geoip2 libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail
  libnginx-mod-stream libnginx-mod-stream-geoip2 nginx nginx-common nginx-core
0 upgraded, 9 newly installed, 0 to remove and 4 not upgraded.
Need to get 645 kB of archives.
After this operation, 2,382 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu hirsute-security/main amd64 nginx-common all 1.18.0-6ubuntu8.2 [38.7 k
B]
Get:2 http://us.archive.ubuntu.com/ubuntu hirsute-security/main amd64 libnginx-mod-http-geoip2 amd64 1.18.0-6ubu
ntu8.2 [11.9 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu hirsute-security/main amd64 libnginx-mod-http-image-filter amd64 1.18.
0-6ubuntu8.2 [15.1 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu hirsute-security/main amd64 libnginx-mod-http-xslt-filter amd64 1.18.0
-6ubuntu8.2 [13.4 kB]
```

        - We can check to verify the installation by using the following command:
            - *nginx -v*

```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ nginx -v
nginx version: nginx/1.18.0 (Ubuntu)
rikeshkarma@pop-os:~$
```

    - Now after installation, we need to enable our firewall using the command:
        - *sudo ufw enable*

```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo ufw enable
Firewall is active and enabled on system startup
rikeshkarma@pop-os:~$
```

    - Now since we have installed our firewall we can list the application configuration that our firewall knows, so for that, we need to use the following command:
        - *sudo ufw app list*

```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo ufw app list
Available applications:
  CUPS
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
rikeshkarma@pop-os:~$
```

        - After using this command we will see all the available applications our Nginx applications - ‘Nginx Full’, ‘Nginx HTTP’ and ‘Nginx HTTPS’.
    - Let’s allow all of them so that all the encrypted and unencrypted traffic is allowed. So for this lets just use the following commands:
        - *sudo ufw allow ‘Nginx Full’*
        - *sudo ufw allow ‘Nginx Http’*
        - *sudo ufw allow ‘Nginx Https’*

```
rikeshkarma@pop-os:~$ sudo ufw app list
Available applications:
  CUPS
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
```

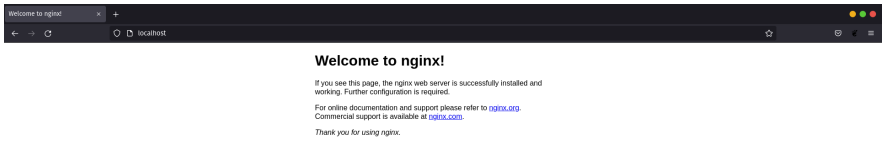
        - Based on whatever kind of traffic we want ot allow we can make sure the application has been enabled.
    - Now we can check the status by typing:
        - *sudo ufw status*

```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
8089                       ALLOW       Anywhere
Nginx Full                 ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
Nginx HTTPS                ALLOW       Anywhere
8089 (v6)                  ALLOW       Anywhere (v6)
Nginx Full (v6)            ALLOW       Anywhere (v6)
Nginx HTTP (v6)            ALLOW       Anywhere (v6)
Nginx HTTPS (v6)           ALLOW       Anywhere (v6)

rikeshkarma@pop-os:~$
```

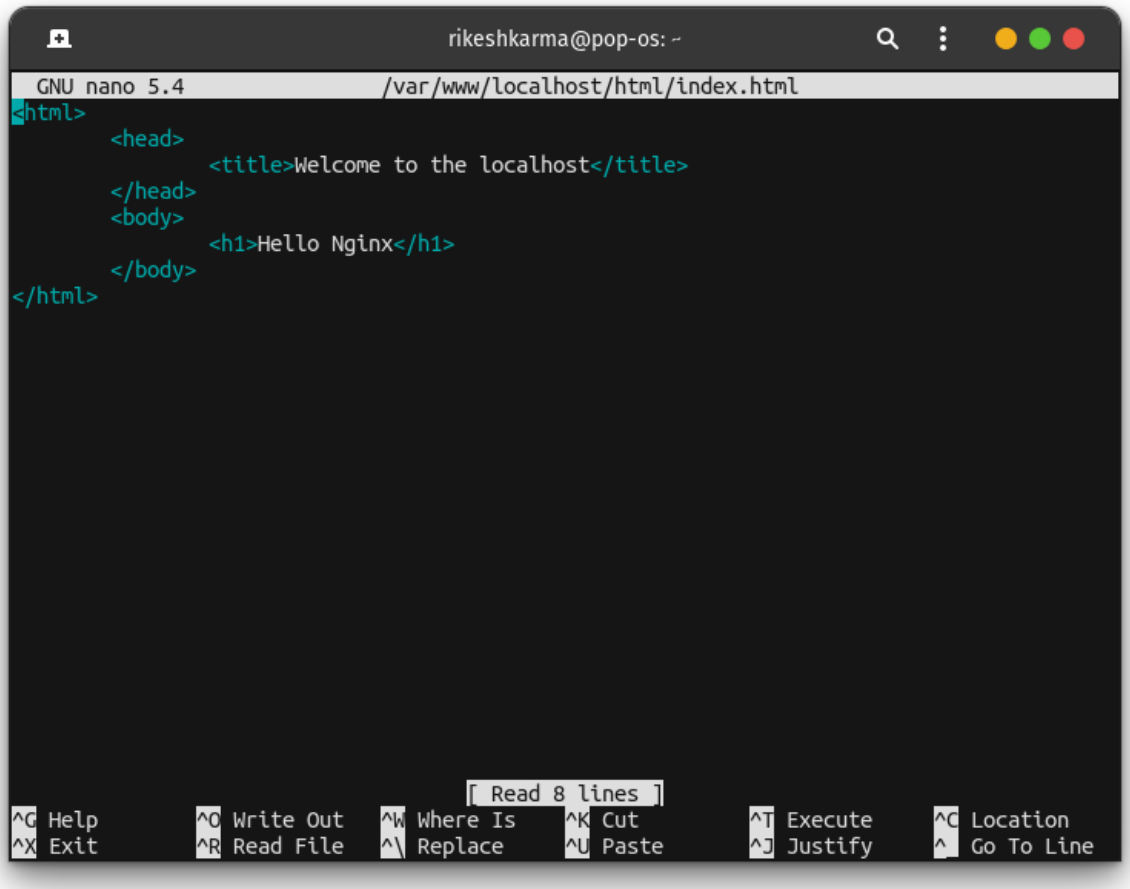
    - We can see that ngnix, http and Https are allowed. This means they are completely fine and they are working.
    - To check if our ngnix server is working or not we can just type in the command:
    - We can check on the browser as well:
        - *sudo systemctl status nginx*

```
rikeshkarma@pop-os: ~
● 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 2021-11-16 18:00:35 +0545; 4min 36s ago
       Docs: man:nginx(8)
    Process: 1151133 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0>
    Process: 1151134 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Main PID: 1151229 (nginx)
      Tasks: 5 (limit: 18973)
     Memory: 5.1M
     CGroup: /system.slice/nginx.service
             ├─1151229 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             ├─1151231 nginx: worker process
             ├─1151232 nginx: worker process
             ├─1151233 nginx: worker process
             └─1151234 nginx: worker process

Nov 16 18:00:35 pop-os systemd[1]: Starting A high performance web server and a reverse proxy server...
Nov 16 18:00:35 pop-os systemd[1]: Started A high performance web server and a reverse proxy server.
~
lines 1-18/18 (END)
```

    - We can check on the browser as well by typing *localhost* on the browser's address bar.



- Now to host a simple index.html with the message “Hello Nginx”.
    - Firstly we need to create a directory for localhost by invoking the command:
        - *sudo mkdir -p /var/www/localhost/html*
    - Then we need to create a new HTML file named index.html for our sample page using the command:
        - *nano /var/www/localhost/html/index.html*
    - Now let’s configure the HTML file to display the message “Hello Nginx”



    - Afterwards need to create *localhost.conf* file inside */etc/nginx/sites-available* using the following command:
        - *sudo nano /etc/nginx/sites-available/localhost.conf*
    - Then we need to add the following lines of code inside the *localhost.conf* file:
        - *server {*
            listen 80;
            listen [::]:80;
            root /var/www/localhost/html;

```
        index.html index.nginx-debian.html;
        server_name localhost;
        location / {
                try_files $uri $uri/ =404;
        }
}
```

- Now we need to enable the file by creating a link that Nginx will read from during the startup, using the following command:
    - *sudo ln -s /etc/nginx/sites-available/localhost/ /etc/nginx/sites-enabled/*

- We can test by using the command:
    - *sudo nginx -t*

- Let's restart Nginx using the command:
    - *sudo systemctl restart nginx*

Finally, let's check if our simple *index.html file is up and running:*

2. What are Nginx header security and its uses? And also implement in the test.conf file.
    - Security Headers on NGINX

        On contrary to Apache-based webservers which use a .htaccess file, Really Simple SSL Pro cannot write security headers directly to your NGINX configuration. NGINX uses an nginx.conf file which is usually located in the /etc/nginx/ folder or a specific site configuration file in the etc/nginx/sites-enabled/ folder. This is outside of the server's public content, therefore Really Simple SSL cannot access it. Don't worry, the security headers can still be used in NGINX.

        The headers can be added via PHP or to the NGINX configuration directly. Do note that adding the headers via PHP can result in issues when using caching. We, therefore, recommend adding the headers to your nginx.conf file. This is something you can do yourself or ask your hosting provider to do for you. Below you will find the correct syntax for each recommended security header.

        The HTTP headers are the additional information passed between the client (browsers) and web servers. The security HTTP headers are the response HTTP headers that the server can add in order to increase the security of HTTP exchange. Each security header serves its own purpose. The HTTP headers in NGINX are split into two parts: the input request headers (headers_in structure) and the output request headers (headers_out structure). There is no such entity as a response, all the data is stored in the same single request structure. The actual response data is constructed from the request data and the headers_out structure fields.

        Nowadays too many data breaches are happening, many websites are hacked due to misconfiguration or lack of protection. These security headers will protect your website from some common attacks like XSS, code injection, clickjacking, etc. Some important security headers are:

        1. <u>HSTS - HTTP Strict Transport Security</u>

            This header instructs a user agent to only use HTTPS connections and it is also declared by Strict-Transport-Security. This will prevent web browsers from accessing web servers over non-HTTPS connections. Currently, all major web browsers support HTTP strict transport security.

            We can implement HSTS in Nginx by adding the following entry in

/etc/nginx/sites-available/test.conf file:
add_header Strict-Transport-Security 'max-age=31536000; includeSubDomains; preload';
It caches this information for the max-age period (typically 31,536,000 seconds, equal to about 1 year). The optional includeSubDomains parameter tells the browser that the HSTS policy also applies to all subdomains of the current domain.

2. <u>X-Frame-Options</u>
   The X-Frame-Options header can help prevent click-jacking attacks. This header indicates whether the web page can be rendered in a frame. X-Frame-Options headers allow you to tell the web browser not to allow embedding of your web pages in a frame. Can be implemented as:
   add_header X-Frame-Options "SAMEORIGIN" always;
   - ➔ SAMEORIGIN – allow your web pages to be displayed in an iframe on the same website
   - ➔ ALLOW-FROM URI – allow your webpages to be embedded in only specific domains/websites
   - ➔ DENY – do not allow any website to embed your webpages in an iframe

3. <u>Content Security Policy</u>
   The Content-Security-Policy header is an improved version of the X-XSS-Protection header and provides an additional layer of security. It is a very powerful header aimed to prevent XSS and data injection attacks. The CSP header is a way of whitelisting the things that your site is allowed to run. This includes images, stylesheets, javascript, inline styles, frames.
   Content-Security-Policy: policy
   Can be implemented as:
   *add_header Content-Security-Policy: "default-src 'self'; font-src *;img-src * data:; script-src *; style-src *; *.youtube.com; *.facebook.com";*

4. <u>X-XSS-Protection</u>
   X-XSS is also known as the Cross-Site Scripting header is used to defend against Cross-Site Scripting attacks.
   Can be implemented as:
   add_header X-XSS-Protection "1; mode=block" always;
   - ➔ X-XSS-Protection: 0: This will disable the filter entirely.
   - ➔ X-XSS-Protection: 1: This will enable the filter but only sanitizes potentially malicious scripts.

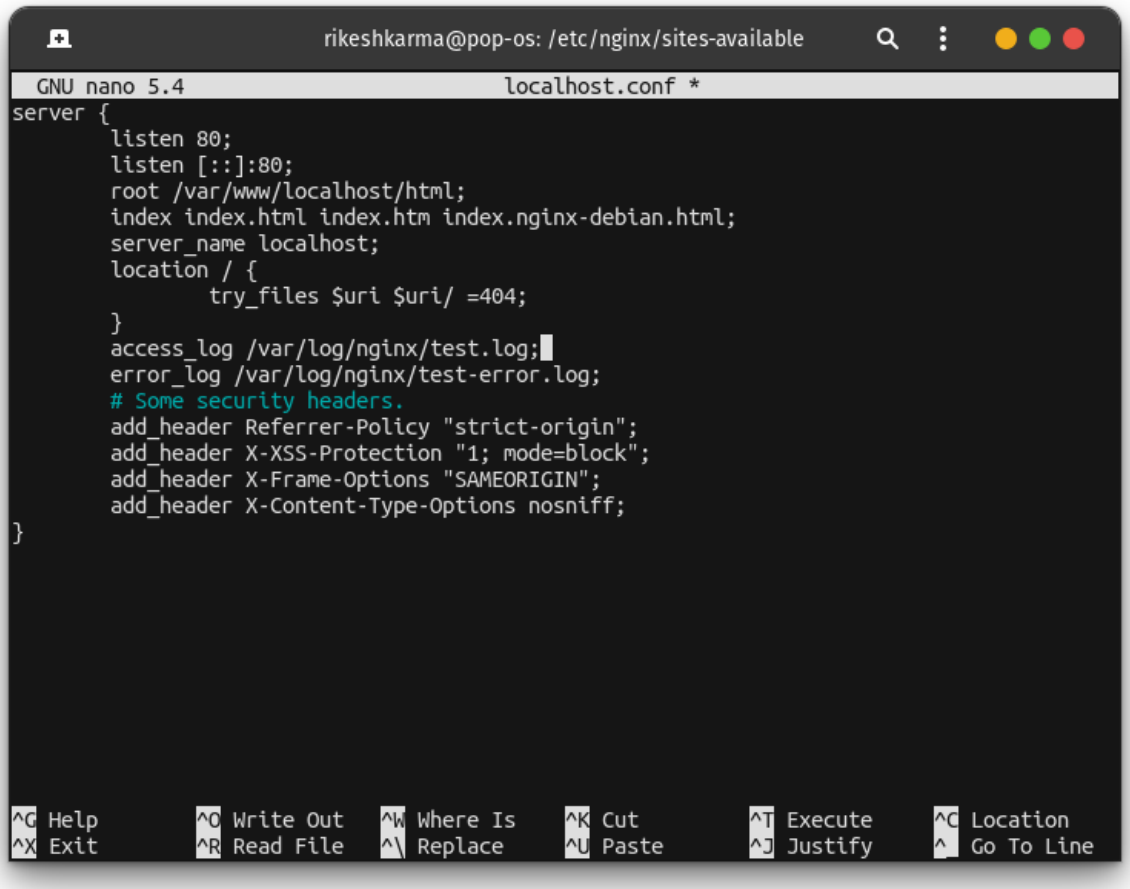
- ➔ X-XSS-Protection: 1; mode=block: This will enable the filter and completely block the page.

5. <u>X-Content-Type-Options</u>
   The x-content-type header is also called "Browser Sniffing Protection" to tell the browser to follow the MIME types are indicated in the header. It is used to prevent web browsers such as Internet Explorer and Google Chrome from sniffing a response away from the declared Content-Type.
   Can be implemented as:
   add_header X-Content-Type-Options "nosniff" always;

6. <u>Feature-Policy</u>
   The HTTP Feature-Policy header provides a mechanism to allow and deny the use of browser features in its own frame, and in content within any iframe elements in the document.
   Syntax:
   Feature-Policy: <directive> <allowlist>
   Can be implemented as:
   add_header Feature-Policy " microphone 'none' ; geolocation 'none' " ;
   Some of the Directives are:
   Camera, midi, accelerometer, fullscreen, etc

   <u>Allowlist values</u>
   - ➢ *: The feature will be allowed in this document, and all nested browsing contexts (iframes) regardless of their origin.
   - ➢ 'self': The feature will be allowed in this document, and in all nested browsing contexts (iframes) of the same origin. The feature is not allowed in cross-origin documents in nested browsing contexts.
   - ➢ 'src': (In an iframe allow attribute only) The feature will be allowed in this iframe, as long as the document loaded into it comes from the same origin as the URL in the iframe's src attribute. Note: The 'src' origin is used in the iframe allow attribute only, and is the default allow list value.
   - ➢ 'none': The feature is disabled in top-level and nested browsing contexts.
   - ➢ <origin(s)>: The feature is allowed for specific origins (for example, https://example.com). Origins should be separated by a space.

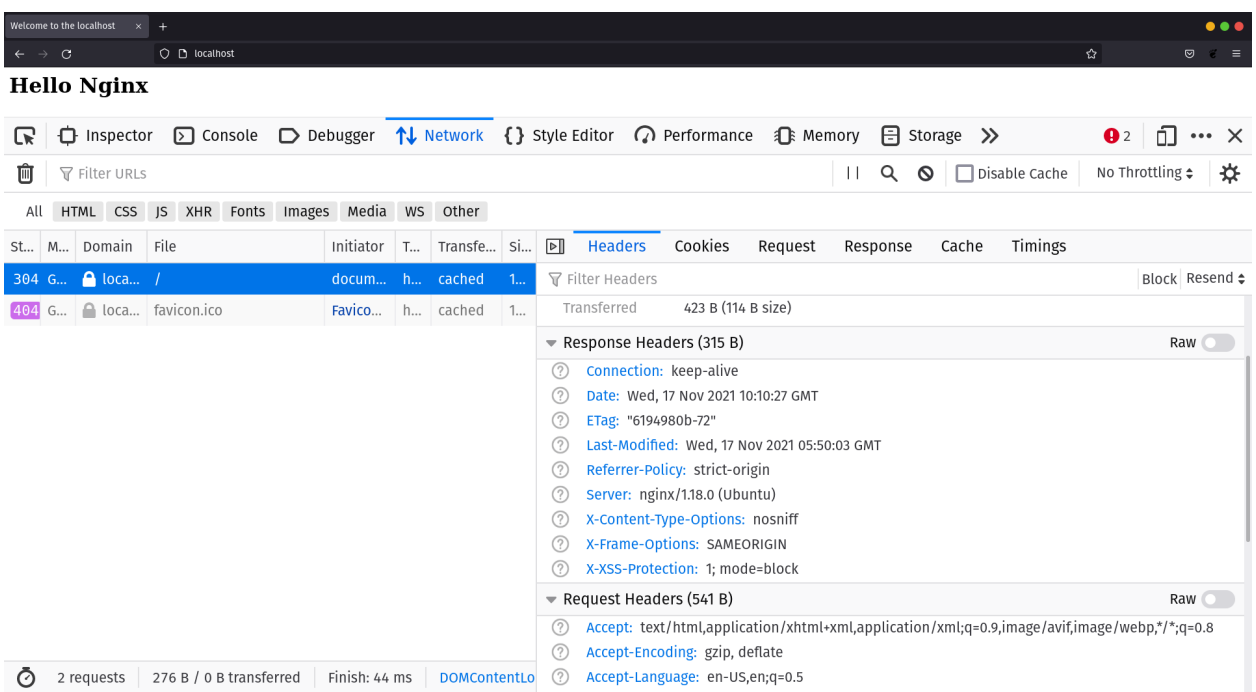
- To implement Nginx Security Headers in the *test.conf* file, we need to add some of these headers into our *localhost.conf file* to implement those securities:
    - Firstly let's navigate to */etc/nginx/sites-available*
    - Then use the following command and add the security headers:
        - *sudo nano localhost.conf*



        - *Added the security headers.*
- Now let's create a link from it to the sites-enabled directory, from which Nginx will read from startup using the following command:
    - *sudo ln -rs /etc/nginx/sites-available/localhost.conf /etc/nginx/sites-enabled/*
- Now let's test:
    - *sudo nginx -t*
- Then restart the Nginx service:
    - *sudo systemctl restart nginx*
    - Let's check the status of Nginx by:
        - *sudo systemctl start nginx*

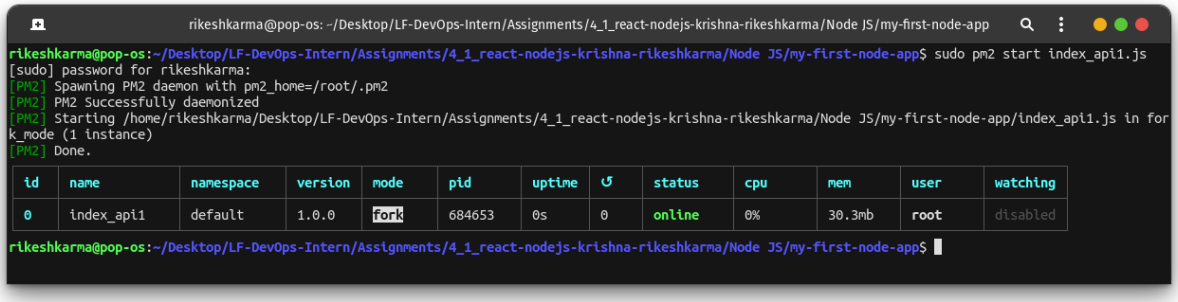
```
rikeshkarma@pop-os: ~
rikeshkarma@pop-os:~$ sudo ln -rs /etc/nginx/sites-available/localhost.conf /etc/nginx/sites-enabled/
rikeshkarma@pop-os:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
rikeshkarma@pop-os:~$ sudo systemctl restart nginx
rikeshkarma@pop-os:~$ sudo 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 Wed 2021-11-17 15:51:17 +0545; 10s ago
       Docs: man:nginx(8)
    Process: 674298 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 674299 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Main PID: 674300 (nginx)
      Tasks: 5 (limit: 18973)
     Memory: 4.6M
     CGroup: /system.slice/nginx.service
             ├─674300 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             ├─674301 nginx: worker process
             ├─674302 nginx: worker process
             ├─674303 nginx: worker process
             └─674304 nginx: worker process

Nov 17 15:51:17 pop-os systemd[1]: nginx.service: Succeeded.
Nov 17 15:51:17 pop-os systemd[1]: Stopped A high performance web server and a reverse proxy server.
Nov 17 15:51:17 pop-os systemd[1]: Starting A high performance web server and a reverse proxy server...
Nov 17 15:51:17 pop-os systemd[1]: Started A high performance web server and a reverse proxy server.
rikeshkarma@pop-os:~$
```

- Now we can check our security headers on our browsers as well:
  - For that let's open our browser and navigate to our localhost.
  - Then on the web browser, Inspect > Network > Headers
  - We can observe the output in the browser with implemented securities.



3. Nginx Reverse proxy all HTTP requests to nodes js API.
    - A proxy server is a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A reverse proxy server is a type of proxy server that directs client requests to the appropriate backend server. It provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers.

    - To reverse proxy Nginx all HTTP requests to node js API, firstly let's start the pm2 tool on the first API during the previous assignment using the following command:
        - *sudo pm2 start index_api1.js*



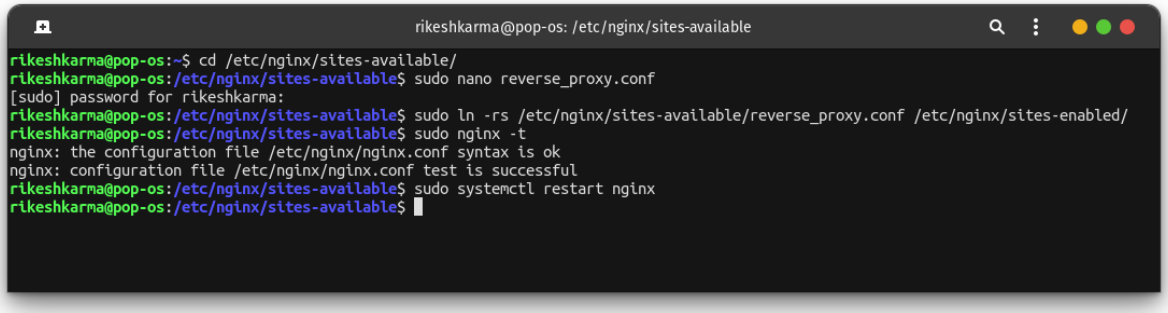
Hello nodejs

    - Now lets create a *reverse_proxy.conf* file inside */etc/nginx/sites-available* using the command:
        - *sudo nano reverse_proxy.conf*
        - And add the below code in the *reverse_proxy.conf* file:

```
server {
    listen 81;
    server_name localhost;

    location / {
        proxy_pass http://localhost:6080;
    }
}
```

- Let's create a soft link to the sites-enabled directory, which helps Nginx to read from the startup and enable this reverse proxy using the command:
  - *sudo ln -rs /etc/nginx/sites-available/reverse_proxy.conf /etc/nginx/sites-enabled/*

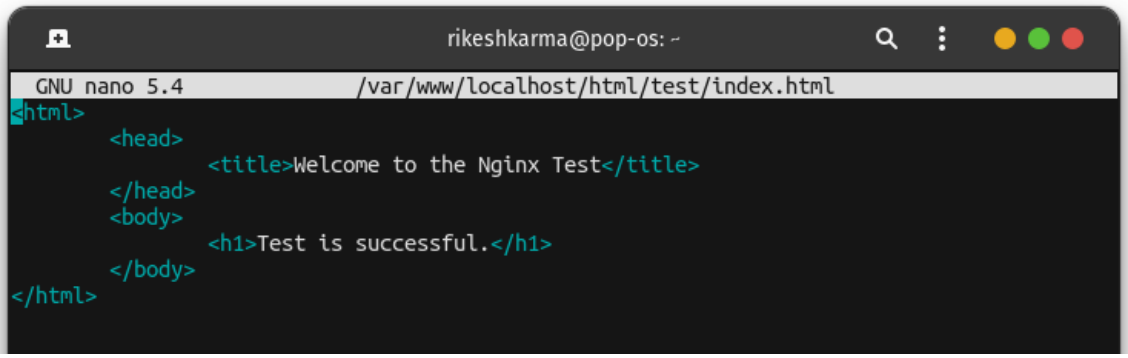
- Then let's test and restart Nginx and check using the following commands:
  - *sudo nginx -t*
  - *sudo systemctl restart nginx*



- Now we can see on the browser that Nginx reverse proxy is up and running.

Hello nodejs

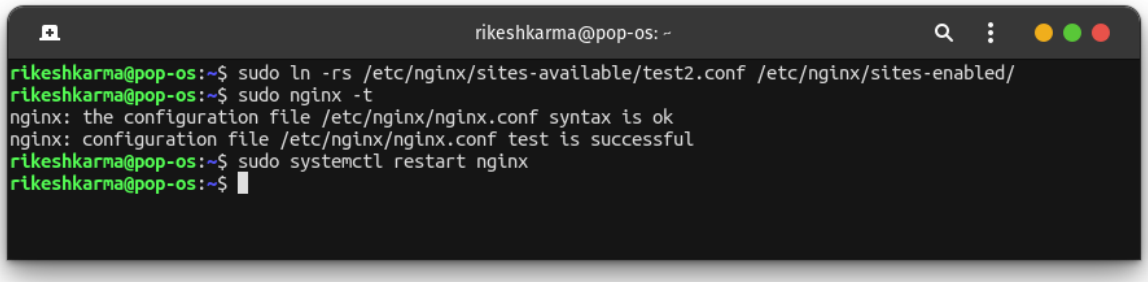
4. Create a test2.conf and listen on port 82 and to “ location /test/” with the message “ Test is successful”.
    - Firstly let’s create a directory for localhost by invoking the following command:
        - *sudo mkdir -p /var/www/localhost/html/test*
    - Then let’s create a sample *index.html* file using the following command:
        - *sudo nano /var/www/localhost/html/test/index.html*
        - And write a simple HTML code inside the HTML file we just created.



    - Now let’s create *test2.conf* file inside */etc/nginx/sites-available* using following command:
        - *sudo nano /etc/nginx/sites-available/test2.conf*
        - *And add the following code in the file ‘test2.conf’*

    - Let’s create a soft link to enable the file, which helps Nginx to read from the startup using the following command:
        - *sudo ln -rs /etc/nginx/sites-available/test2.conf /etc/nginx/sites-enabled/*

- *Now let's test and restart Nginx using the following command:*
  - *sudo nginx -t*
  - *sudo systemctl restart nginx*

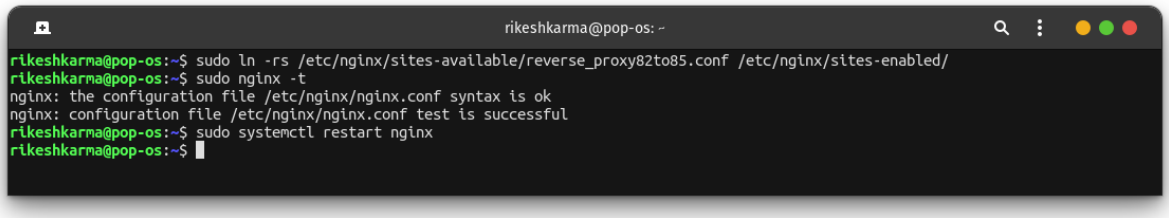


- We can observe that we are getting the message *"Test is successful"* with *localhost:82/test*

Test is successful.

5. Reverse proxy all HTTP traffic of port 82 to port 85.
    - Firstly let's create a *reverse_proxy82to85.conf* inside */etc/nginx/sites-available using following command:*
        - *sudo nano /etc/nginx/sites-available/reverse_proxy82to85.conf*
    - Then add the following on the conf file we just created.

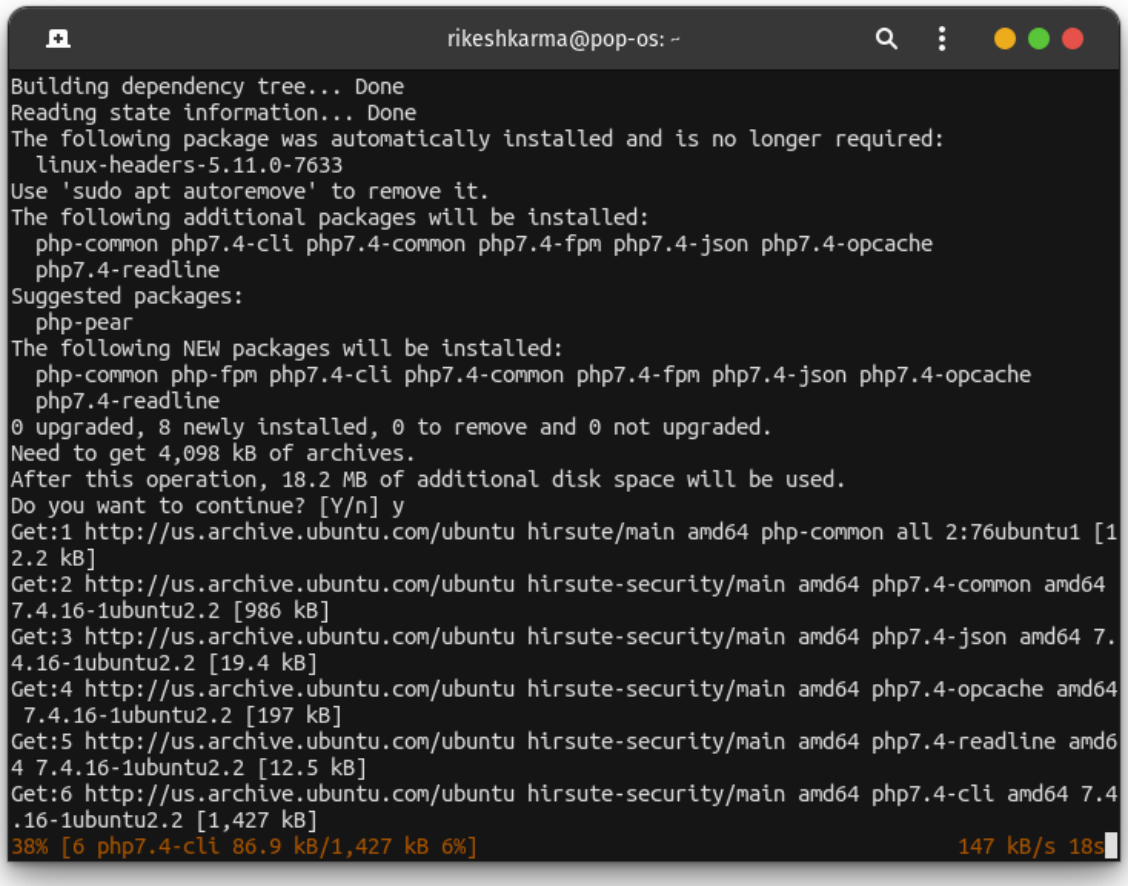
    - Let's create a soft link to the sites-enabled directory, which helps Nginx to read from the startup using the following command:
        - *sudo ln -rs /etc/nginx/sites-available/reverse_proxy82to85.conf /etc/nginx/sites-enabled/*
    - Now let's test and restart Nginx using the following command:
        - *sudo nginx -t*
        - *sudo systemctl restart nginx*



    - *We can observe "Test is successful." at port 85.*

Test is successful.

6. Install LEMP stack (avoid installing MySQL) and open info.php on port 80 and print message info.php.
    - A software stack is a set of software tools bundled together. LEMP stands for Linux, Ngnix, MySQL, and PHP, all of which are open source and free to use. It is the most common software stack that powers dynamic websites and web applications. Linux is the operating system; Ngnix is the web server; MySQL is the database server and PHP is the server-side scripting language responsible for generating dynamic web pages.
    - For our task, we don't need to install MySQL as per the question.
    - Our operating system is already Linux so we don't have to install one and we have already installed Nginx previously, so this leaves us to installing PHP only.
    - To install PHP on our system we can use the following command in our terminal:
        - *sudo apt install php-fpm*



    - Now we need to create a new directory for the localhost, for this, we can invoke the following command in the terminal:
        - sudo mkdir /var/www/php
    - Then create simple *info.php* page using the following command:
        - sudo nano /var/www/php/info.php

- Add the following lines in the file we just created.

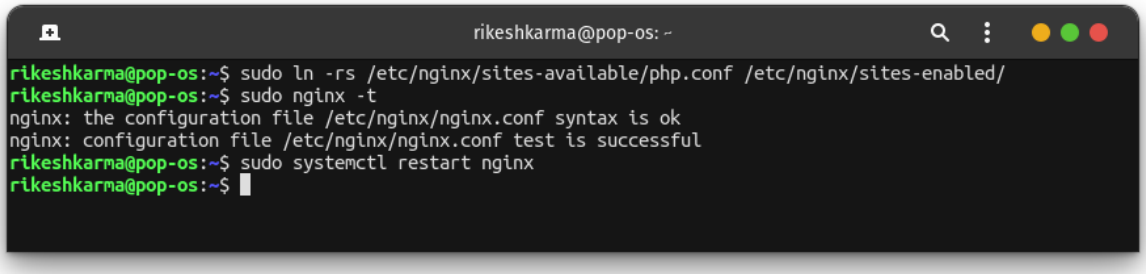
```
rikeshkarma@pop-os: ~
  GNU nano 5.4                    /var/www/php/info.php *
<?php
php info();
?>
```

- We need to create *php.conf* file inside */etc/ngnix/sites-available* using the following command:
  - *sudo nano /etc/nginx/sites-available/php.conf*
  - And add the following lines of code in the *php.conf* file:
  - As port 80 has already been used, lets use port 90 for this task.

```
rikeshkarma@pop-os: ~
  GNU nano 5.4                /etc/nginx/sites-available/php.conf
server {
        listen 90;
        root /var/www/php;
        index index.php index.html index.htm index.nginx-debian.html;

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

        location ~ \.php$ {
                include snippets/fastcgi-php.conf;
                fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        }

        location ~ /\.ht {
                deny all;
        }
}

                                [ Read 18 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut      ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste    ^J Justify    ^_ Go To Line
```

- Let's create a soft link, which helps Nginx to read from the startup using the following command:
  - *sudo ln -rs /etc/nginx/sites-available/php.conf /etc/nginx/sites-enabled/*

- Now let's test and restart Nginx using the following command:
  - *sudo nginx -t*
  - *sudo systemctl restart nginx*



- We can now observe info.php.