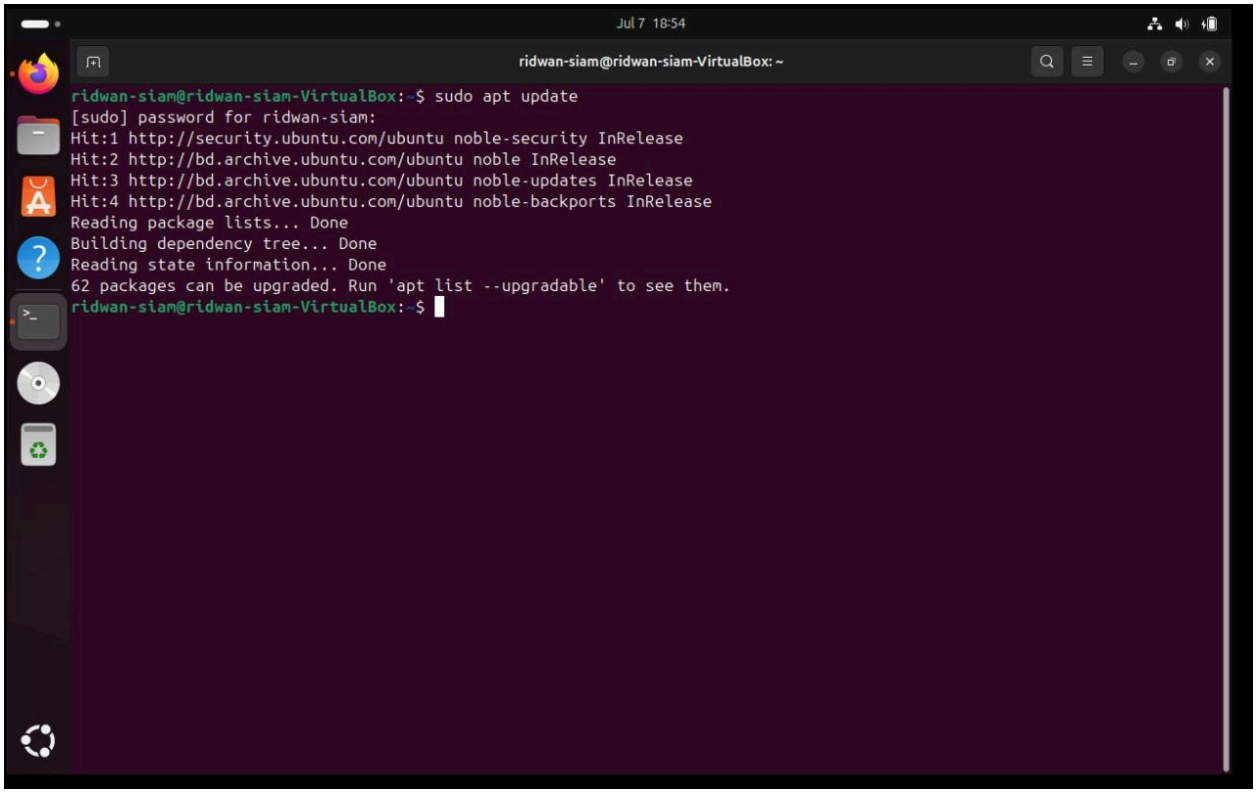
# Task-1: Setting up an Apache web server

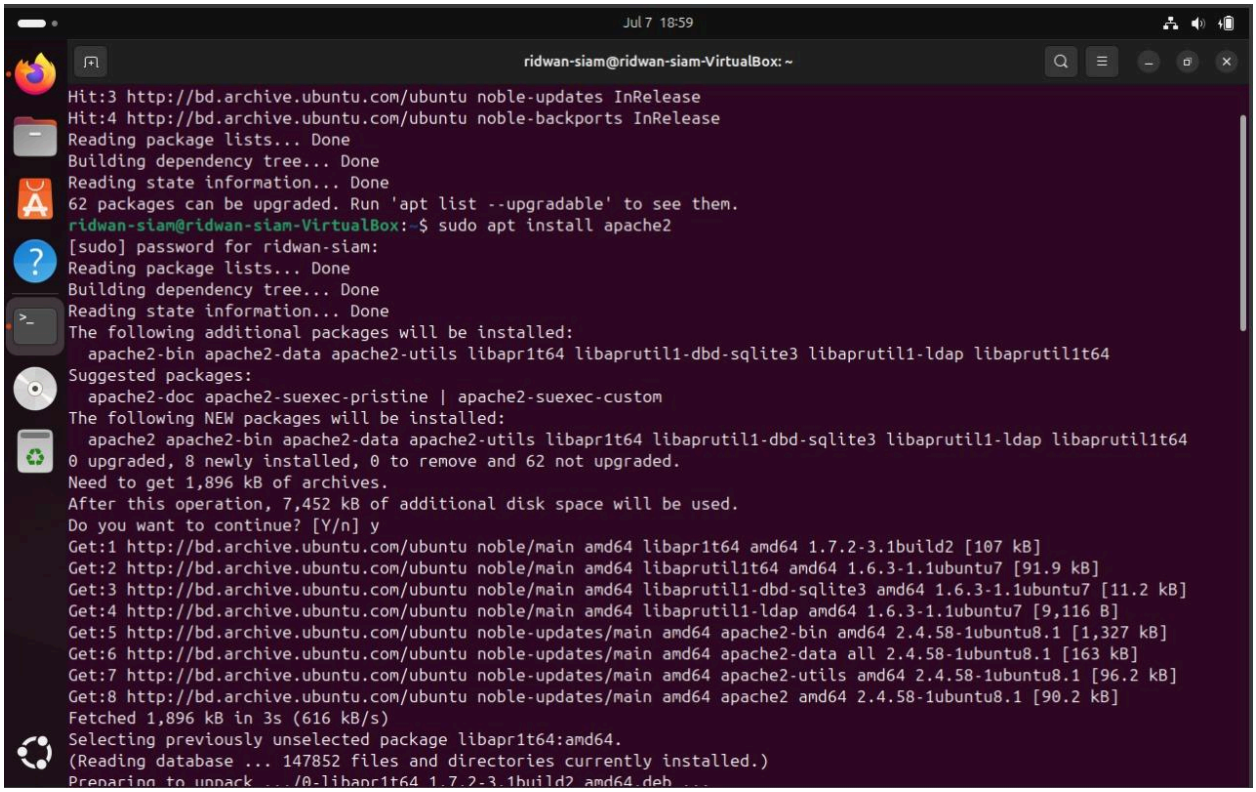
## Step 1 — Installing Apache

Apache is available within Ubuntu's default software repositories, making it possible to install it using conventional package management tools.

Let's begin by updating the local package index to reflect the latest upstream changes. If apt is not recognised as a command, try apt-get instead of apt.
**sudo apt update**.




Then, install the apache2 package:
**sudo apt install apache2**
After confirming the installation, apt will install Apache and all required dependencies




## Step 2 — Adjusting the Firewall

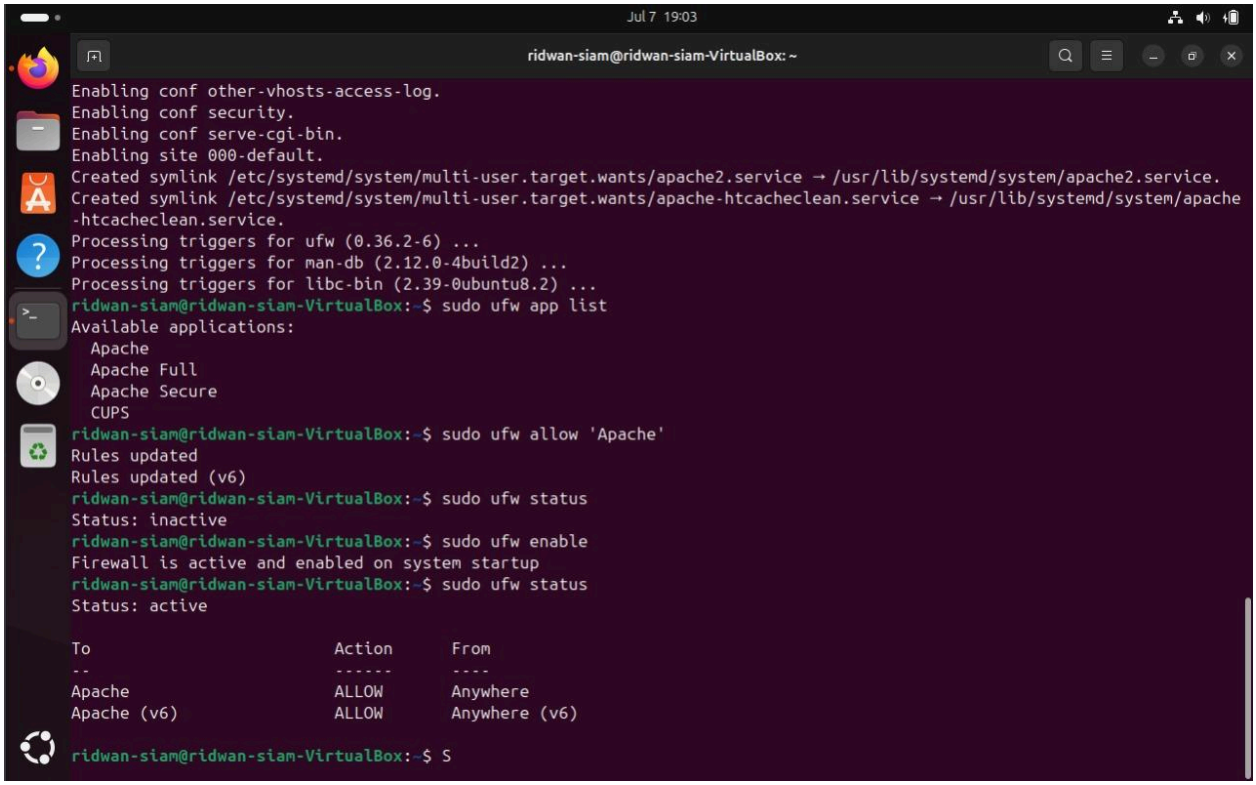
Before testing Apache, it's necessary to modify the firewall settings to allow outside access to the default web ports. This is necessary if you try to access your web site from a separate machine. Assuming that you followed the instructions in the prerequisites, you should have a UFW firewall configured to restrict access to your server.

List the ufw application profiles by typing:
**sudo ufw app list**

It is recommended that you enable the most restrictive profile that will still allow the traffic you've configured. Since we haven't configured SSL for our server yet in this guide, we will only need to allow traffic on port 80:
**sudo ufw allow 'Apache'**

You can verify the change by typing:
**sudo ufw status**
**sudo ufw enable**

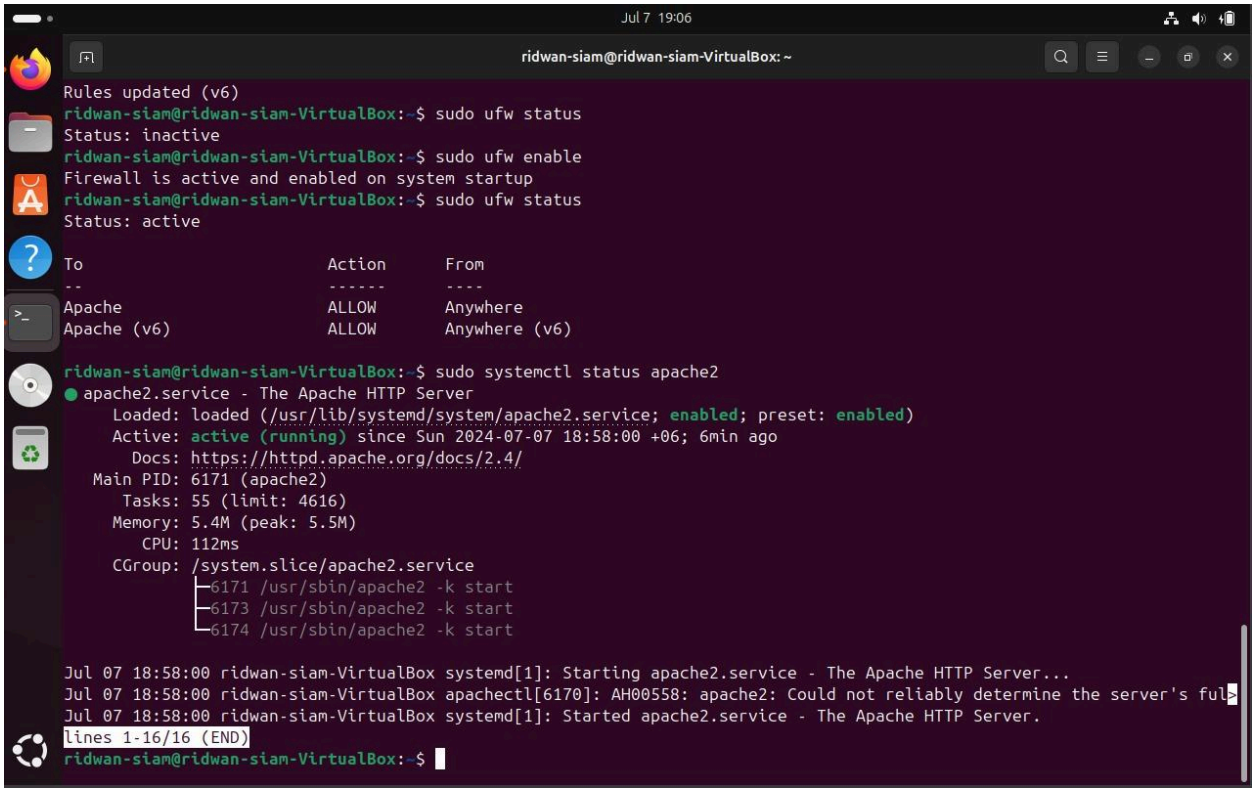
As you can see, the profile has been activated to allow access to the web server.




## Step 3 —Checking your Web Server

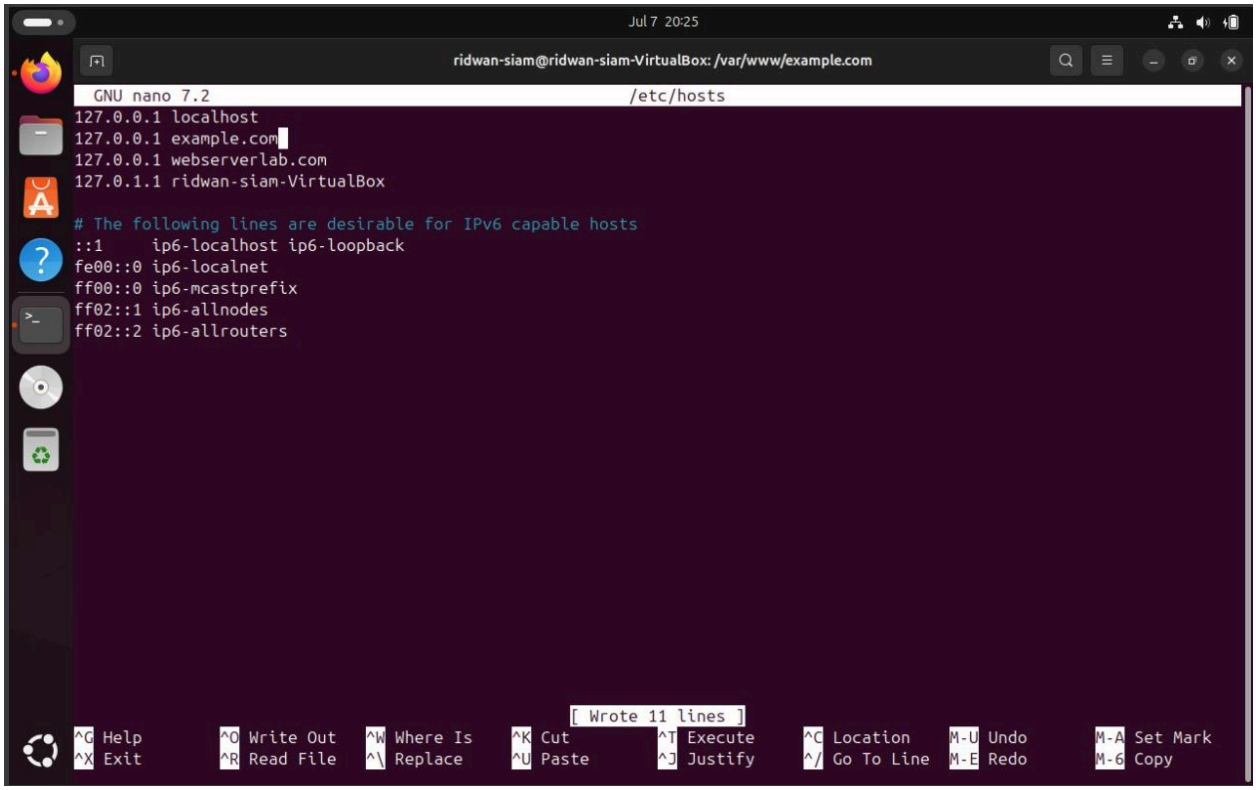
At the end of the installation process, Ubuntu 18.04 starts Apache. The web server should already be up and running.

Check with the systemd init system to make sure the service is running by typing:
**sudo systemctl status apache2**

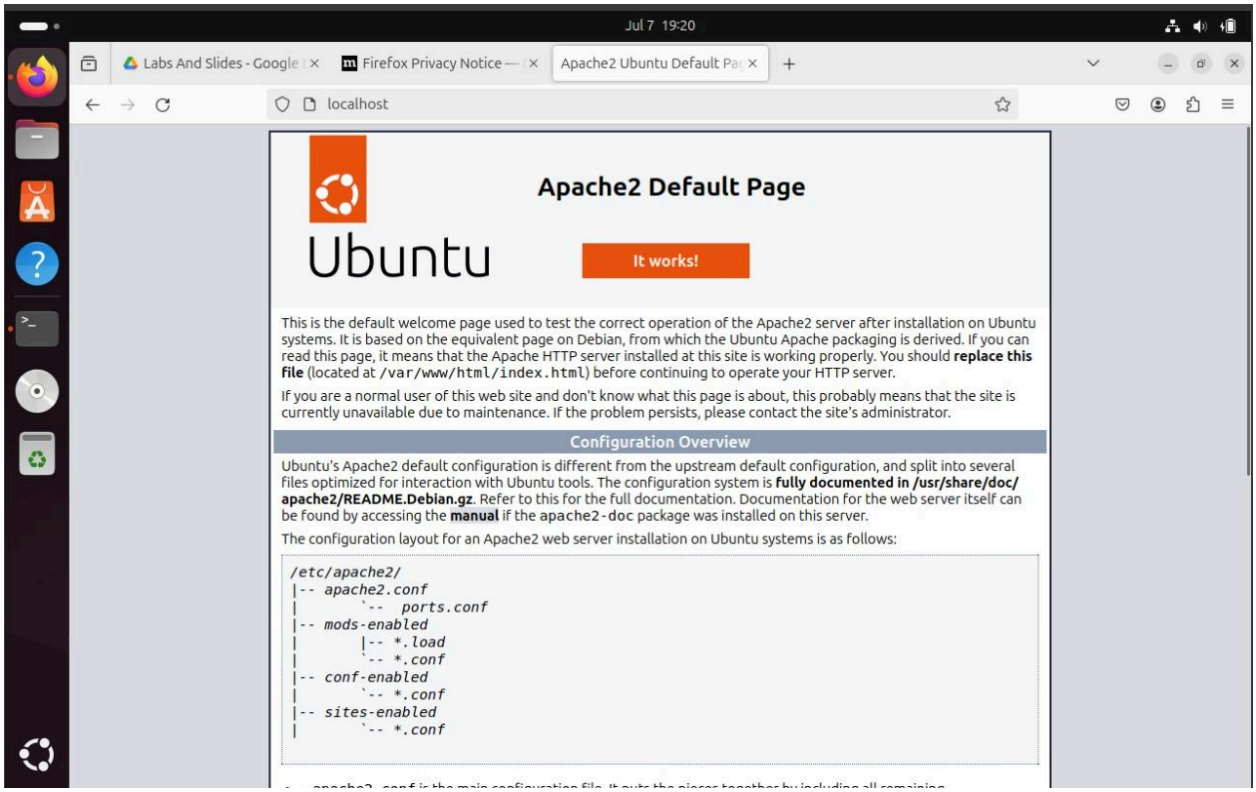



You can access the default Apache landing page to confirm that the software is running properly through your IP address or by just typing localhost (127.0.0.1) in the browser. Let us use webserverlab.com as our domain name. To get our computers recognise this domain name, let us add the following entry to /etc/hosts; this entry basically maps the domain name webserverlab.com to our localhost (i.e., 127.0.0.1):

• 127.0.0.1 webserverlab.com




Now, to check the installation of Apache, enter this domain or its IP address into your browser's address bar:
http://webserverlab.com or http://localhost or http://127.0.0.1 or [http://ip_address](http://ip_address)




## Step 2 — Setting up a single virtual host

Create the directory for example.com as follows, using the -p flag to create any necessary parent directories:

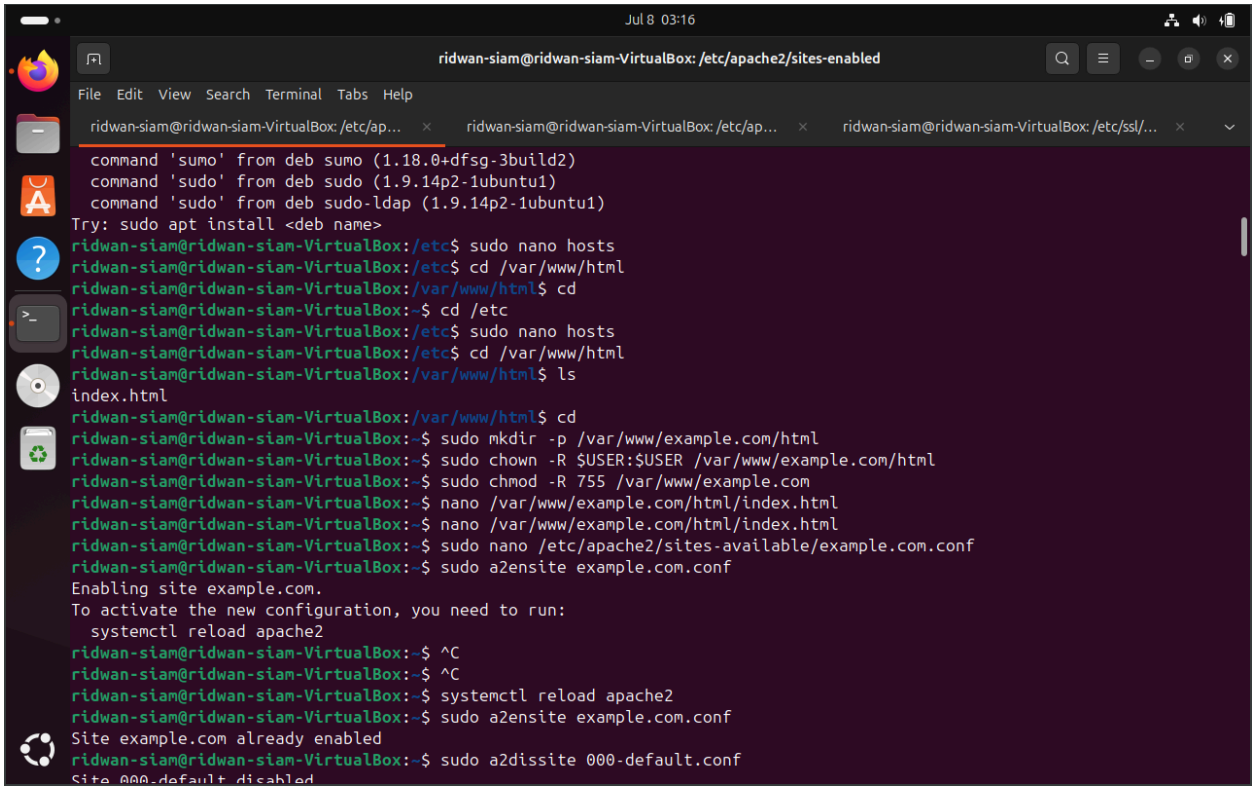
**sudo mkdir -p /var/www/example.com/html**

Next, assign ownership of the directory with the $USER environment variable:

**sudo chown -R $USER:$USER /var/www/example.com/html**

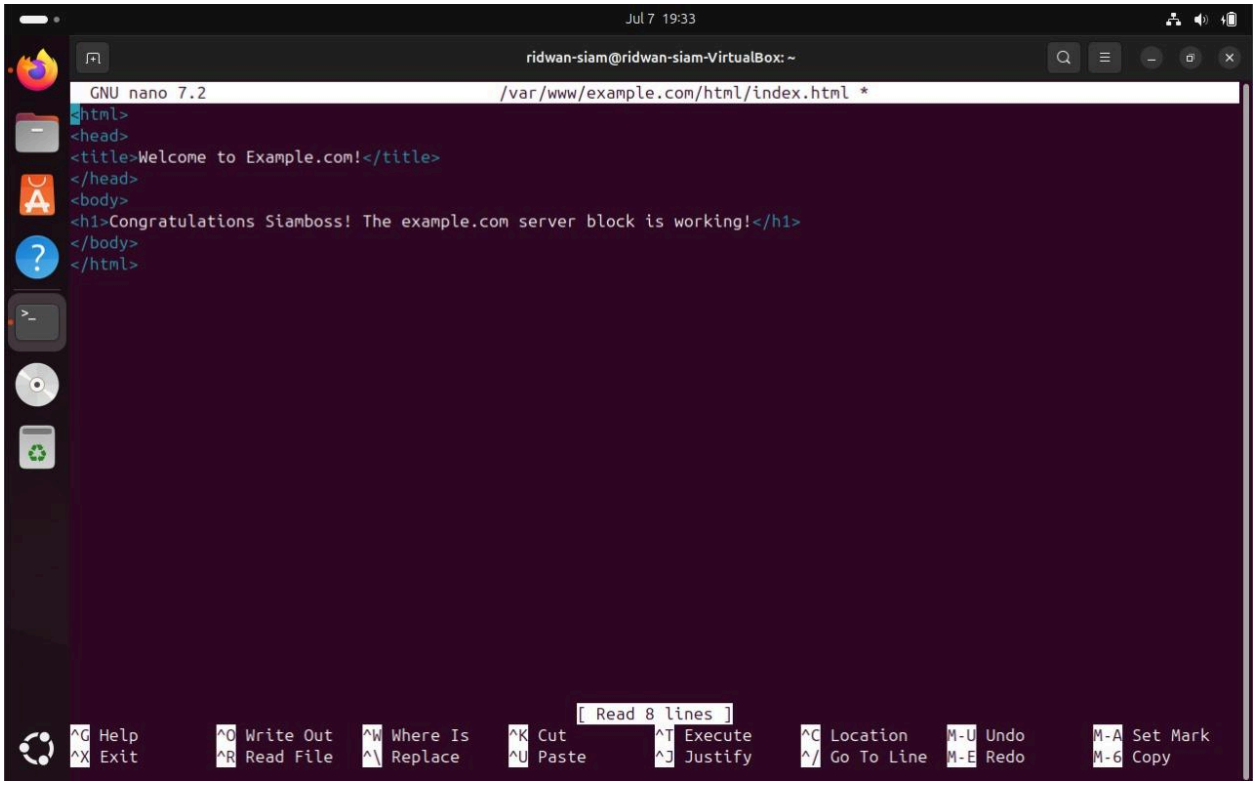
The permissions of your web roots should be correct if you haven't modified your unmask value, but you can make sure by typing:

**sudo chmod -R 755 /var/www/example.com**




Next, create a sample index.html page using nano or your favorite editor:

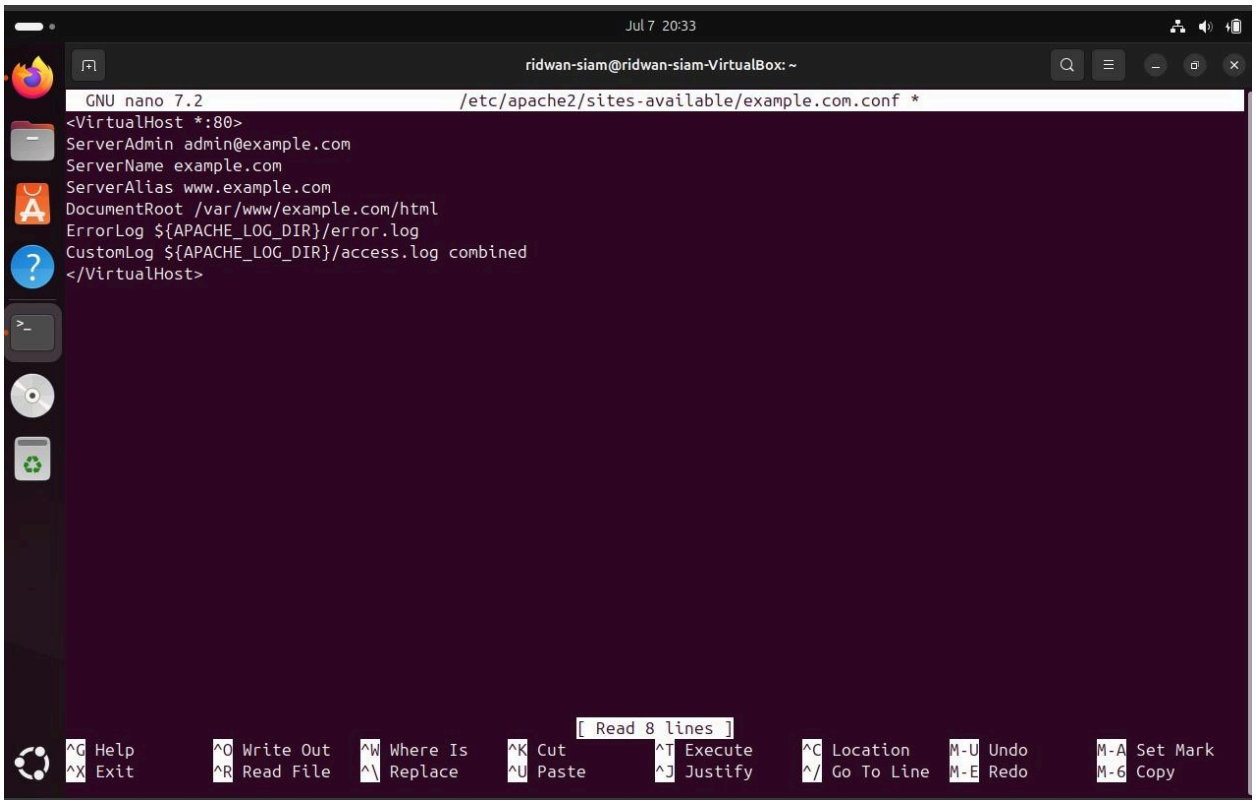
**nano /var/www/example.com/html/index.html**



Save and close the file when you are finished.

In order for Apache to serve this content, it's necessary to create a virtual host file with the correct directives. Instead of modifying the default configuration file located at /etc/apache2/sites-available/000-default.conf directly, let's make a new one at /etc/apache2/sites-available/example.com.conf:

**sudo nano /etc/apache2/sites-available/example.com.conf**




Paste in the following configuration block, which is similar to the default, but updated for our new directory and domain name:
Notice that we've updated the DocumentRoot to our new directory and ServerAdmin to an email that the example.com site administrator can access. We've also added two directives: ServerName, which establishes the base domain that should match for this virtual host definition, and ServerAlias, which defines further names that should match as if they were the base name.

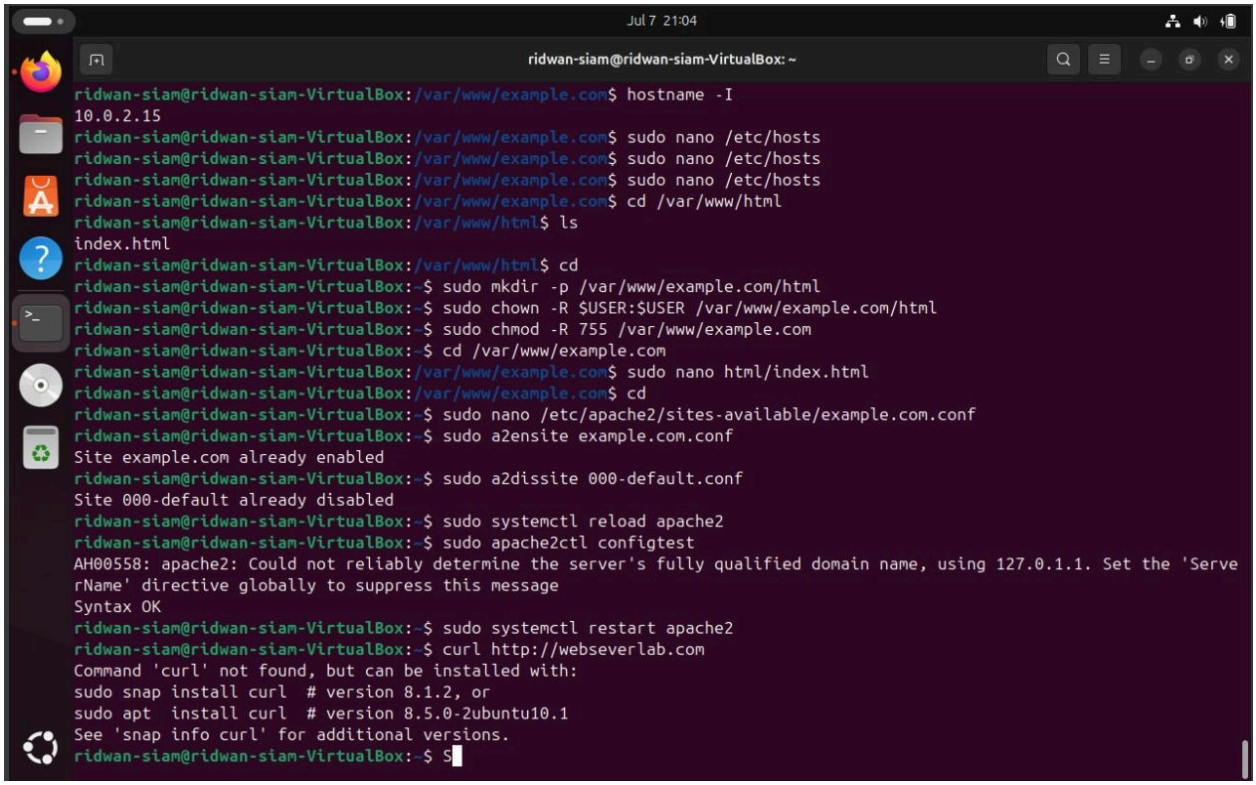
Save and close the file when you are finished.
Let's enable the file with the a2ensite tool:

**sudo a2ensite example.com.conf**

Disable the default site defined in 000-default.conf:
**sudo a2dissite 000-default.conf**

Next, let's test for configuration errors:
**sudo apache2ctl configtest**

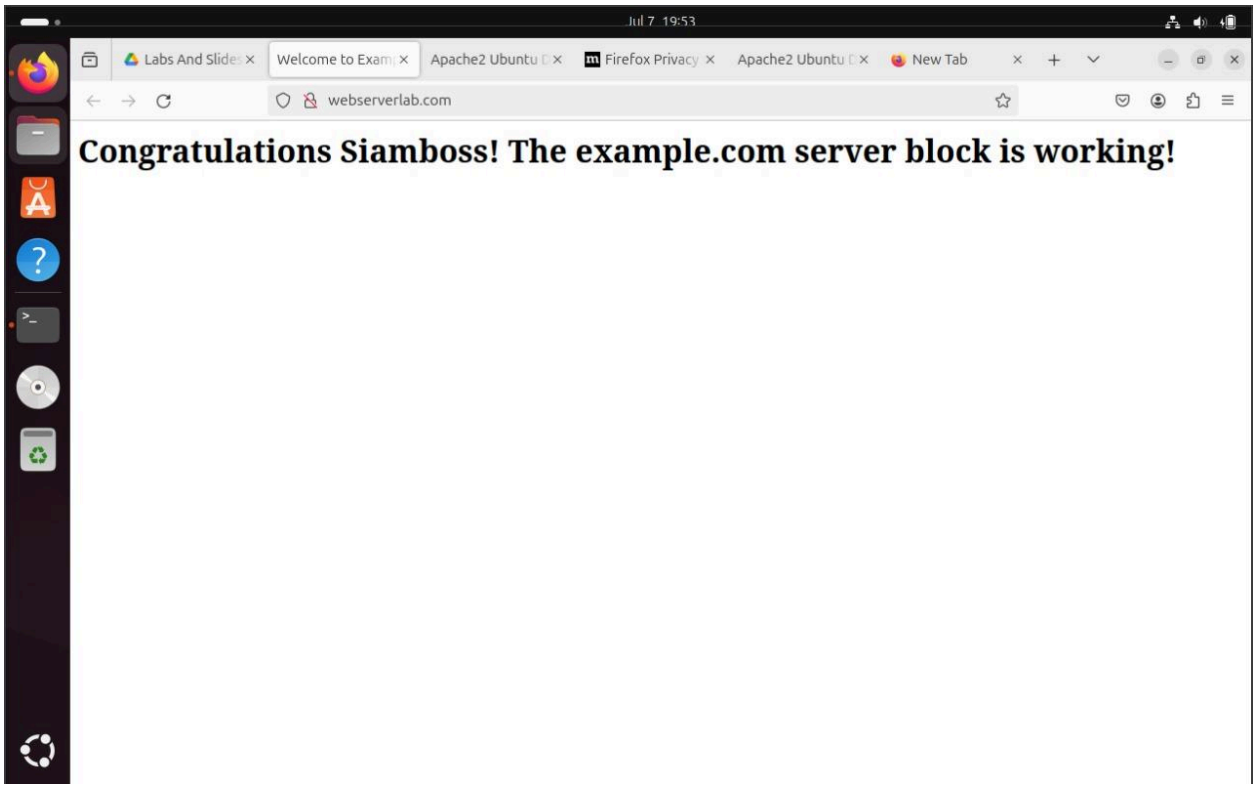
If you see a “Syntax OK” output, then it’s properly configured.




Restart Apache to implement your changes:
**sudo systemctl restart apache2**

Apache should now be serving your domain name. Now issue the following command:
**sudo a2ensite example.com.conf**
Restart Apache to implement your changes:
**sudo systemctl restart apache2**

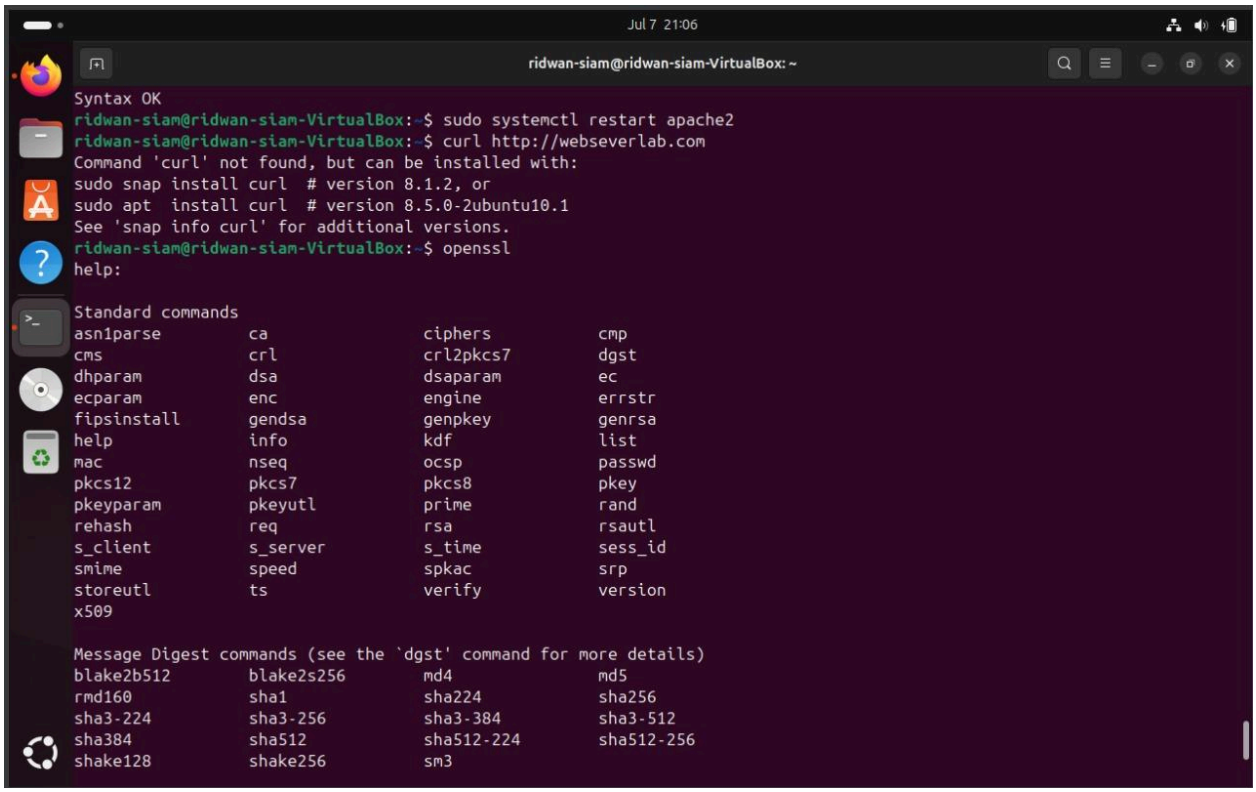
Try navigating to http://webserverlab.com, observe what happens. Think about what is happening. Try to navigate to http://127.0.0.1. What happened and why?


Jul 7 19:53
Labs And Slides
Welcome to Exam
Apache2 Ubuntu D
Firefox Privacy
Apache2 Ubuntu D
New Tab
webserverlab.com
Congratulations Siamboss! The example.com server block is working!

# Lab 5: Securing Apache Web Server

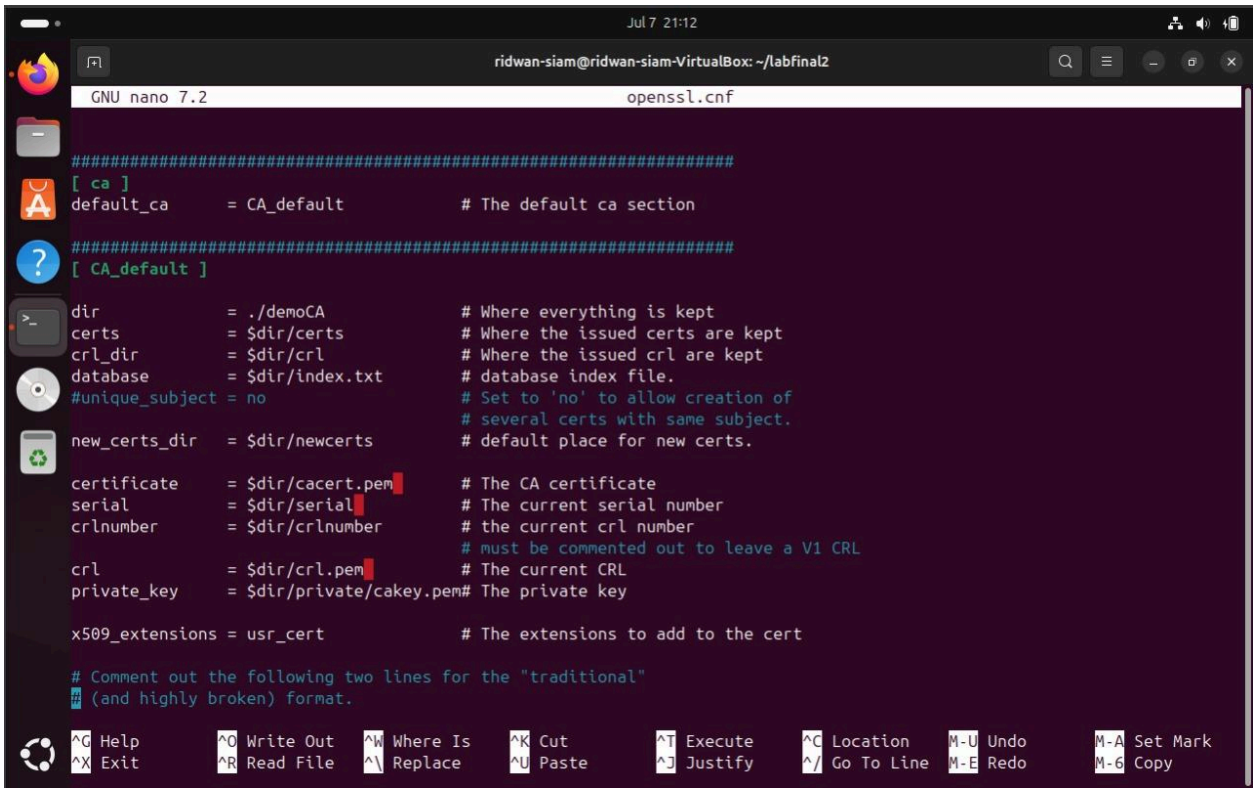
### Task-1: Becoming a certificate authority

In this lab, you will need to create digital certificates, but you will not be going to pay to any commercial CA. You will become a root CA, and then use this CA to issue certificate for others (e.g. servers). In this task, you will make yourself a root CA, and generate a certificate for this CA. Unlike other certificates, which are usually signed by another CA, the root CA's certificates are self-signed. Root CA's certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on certificate-based security. Root CA's certificates are unconditionally trusted.
For this, you will use OpenSSL which are already familiar with from Lab-3.

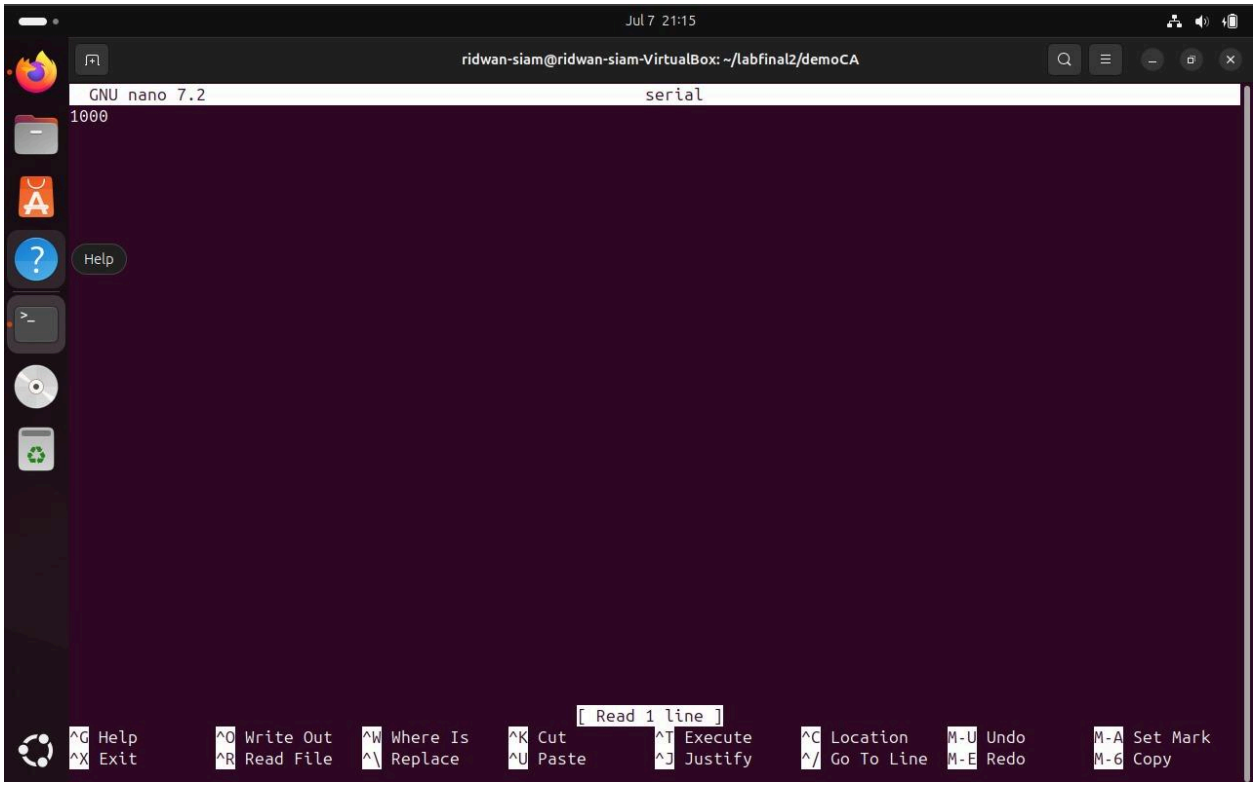



To start this task, create a folder for this task and cd into it. In this folder, you will need to create a particular configuration file as discussed below.

The Configuration File openssl.conf: In order to use OpenSSL to create certificates, you have to have a configuration file. The configuration file usually has an extension .cnf. It is used by three OpenSSL commands: ca, req and x509. The manual page of openssl.conf can be found using Google search. You can also get a copy of the configuration file from /usr/lib/ssl/openssl.cnf. After copying this file into your current directory, you need to create several sub-directories as specified in the configuration file (look at the [CA default] section):

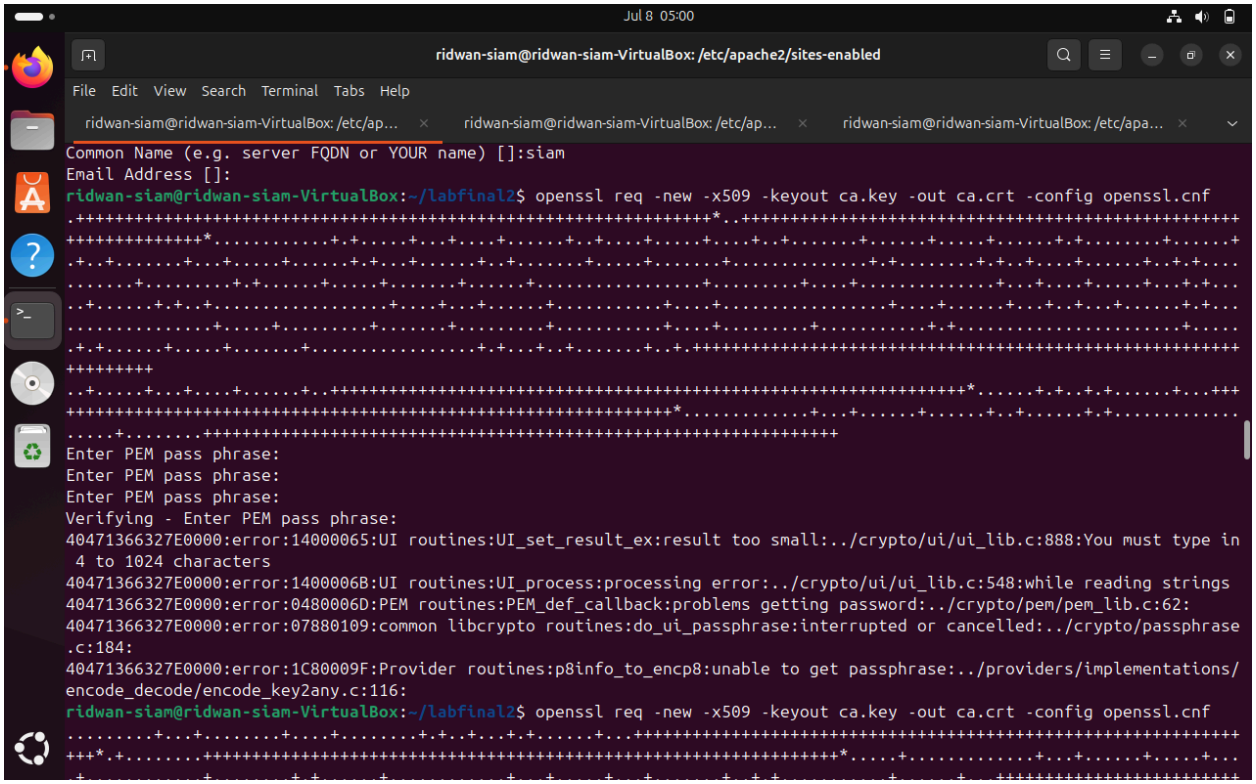



For the index.txt file, simply create an empty file. For the serial file, put a single number in string format (**e.g. 1000**) in the file. Once you have set up the configuration file openssl.cnf, you can create and issue certificates.




Certificate Authority (CA): As described before, you need to generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. You can run the following command to generate the self-signed certificate for the CA:

**openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf**




You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. The output of the command are stored in two files: ca.key and ca.crt. The file ca.key contains the CA's private key, while ca.crt contains the public-key certificate.

Creating a certificate for example.com

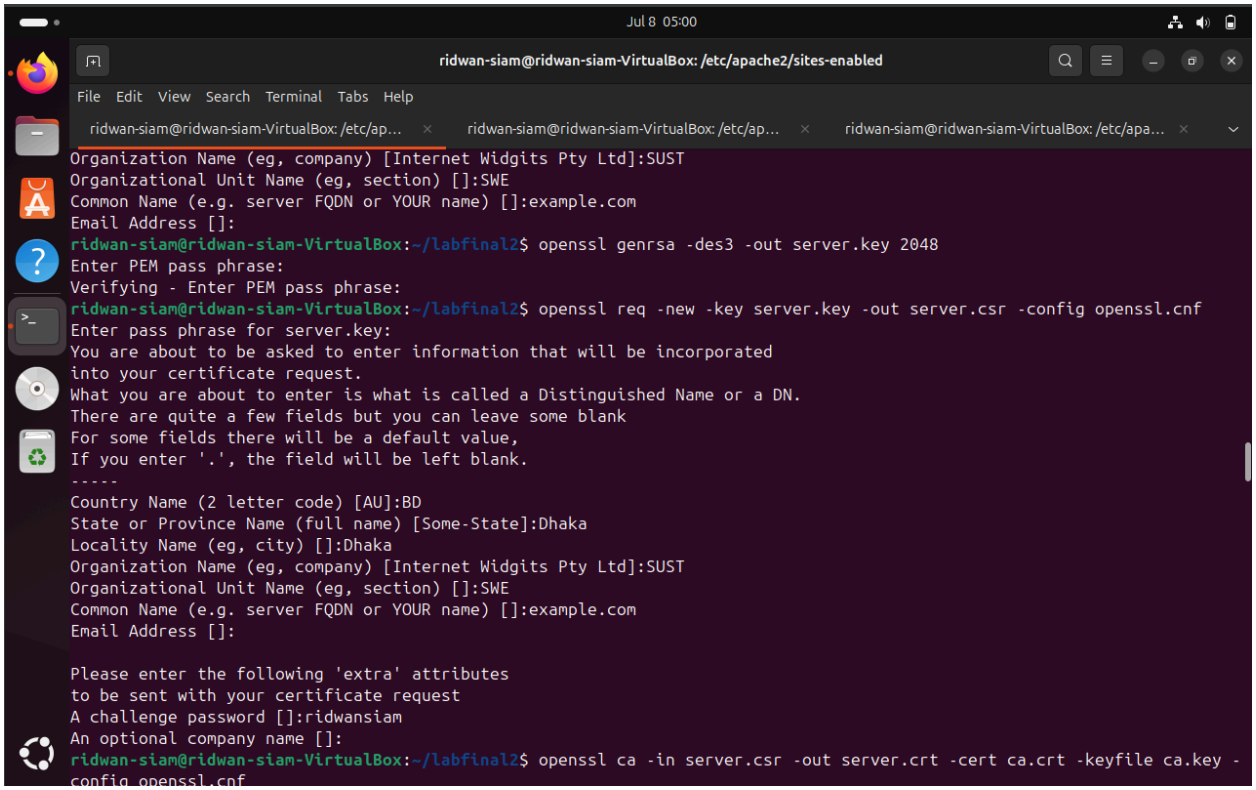
After becoming a root CA, you are ready to sign digital certificates for our customers. Our first customer is a company called example.com. For this company to get a digital certificate from a CA, it needs to go through three steps.

Step 1: Generate public/private key pair. The company needs to first create its own public/private key pair. You can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys. The keys will be stored in the file server.key:

**openssl genrsa -des3 -out server.key 2048**

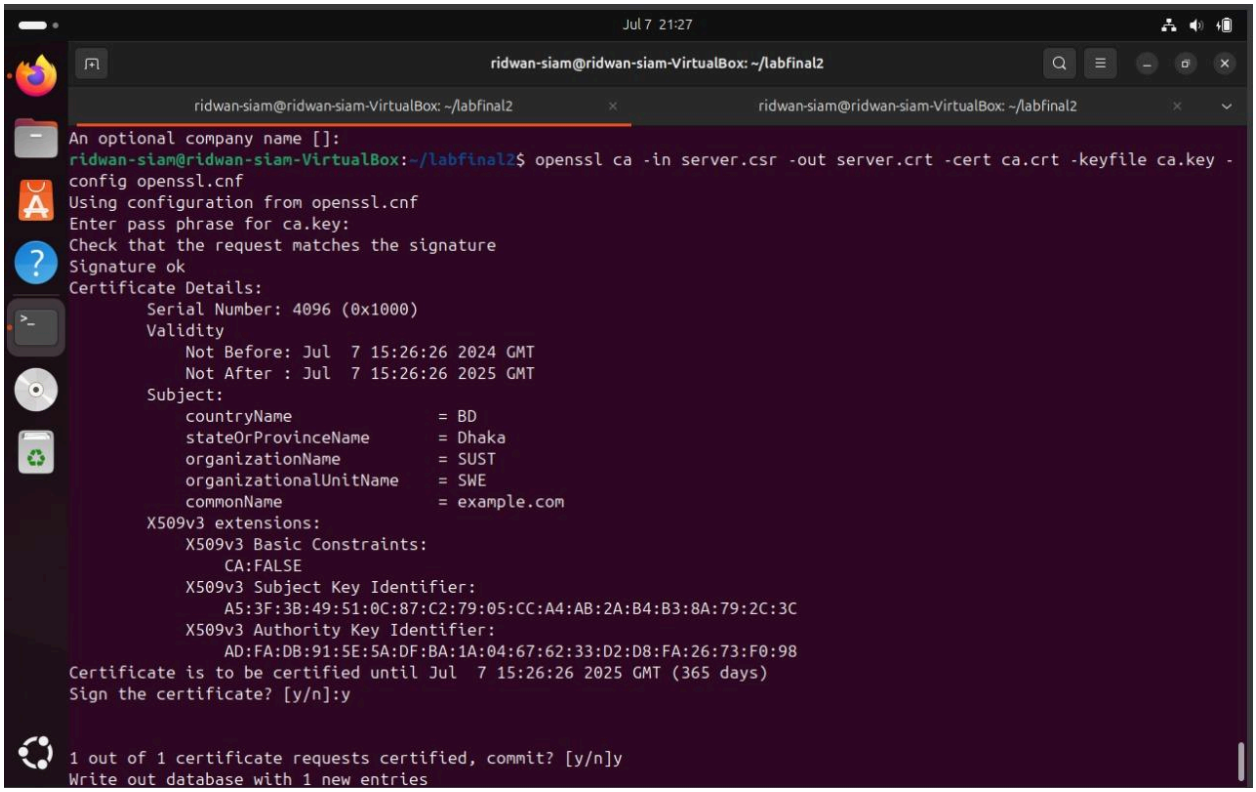
Step 2: Generate a Certificate Signing Request (CSR). Once the company has the key file, it should generates a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). Please use example.com as the common name of the certificate request.

**openssl req -new -key server.key -out server.csr -config openssl.cnf**




Step 3: Generating Certificates. The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, you will use our own trusted CA to generate certificates:

**openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf**




If OpenSSL refuses to generate certificates, it is very likely that the names in your requests do not match with those of CA. Fix this and re-issue the above command.
Next, let us launch a simple web server with the certificate generated in the previous task. OpenSSL allows us to start a simple web server using the s_server command. Use the following steps:

Step 1: Combine the secret key and certificate into one file

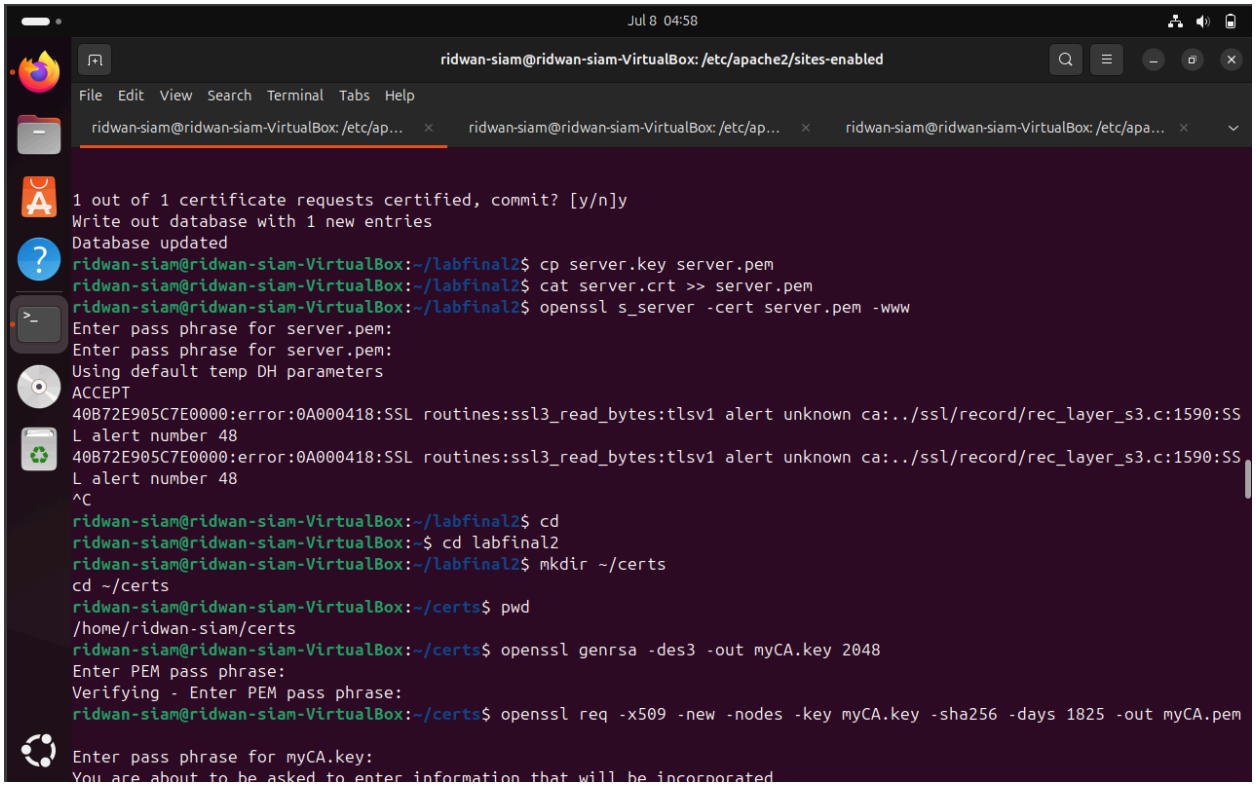
**cp server.key server.pem**

**cat server.crt >> server.pem**

Step 2: Launch the web server using server.pem

**openssl s_server -cert server.pem -www**

By default, the server will listen on port 4433. You can alter that using the -accept option. Now, you can access the server using the following URL: https://example.com:4433/. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following: “example.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown”.




Had this certificate been assigned by VeriSign, you will not have such an error message, because VeriSign's certificate is very likely preloaded into Firefox's certificate repository already. Unfortunately, the certificate of example.com is signed by our own CA (i.e., using ca.crt), and this CA is not recognized by Firefox. There are two ways to get Firefox to accept our CA's self-signed certificate.

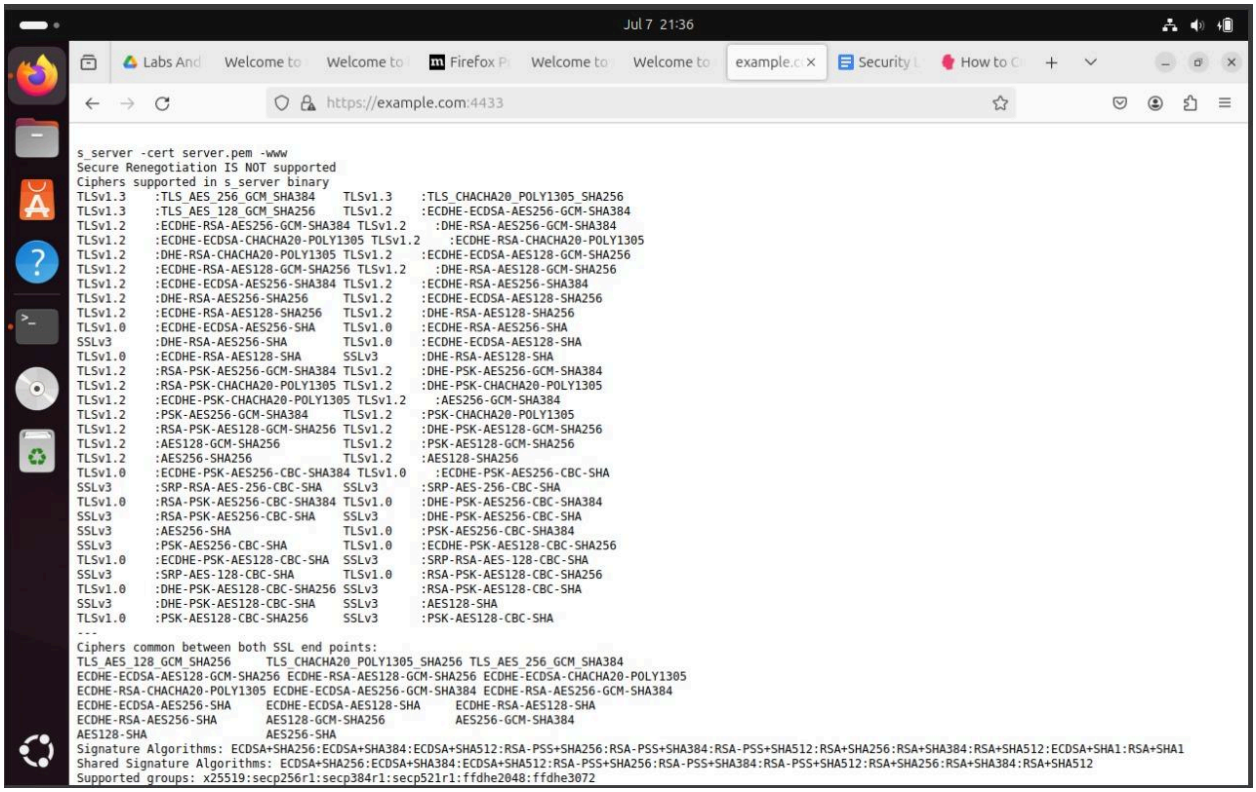
You can request Mozilla to include our CA's certificate in its Firefox software, so everybody using Firefox can recognize our CA. This is how the real CAs, such as VeriSign, get their certificates into Firefox. Unfortunately, our own CA does not have a large enough market for Mozilla to include our certificate, so you will not pursue this direction.

Load ca.crt into Firefox: You can manually add our CA's certificate to the Firefox browser by clicking the following menu sequence:

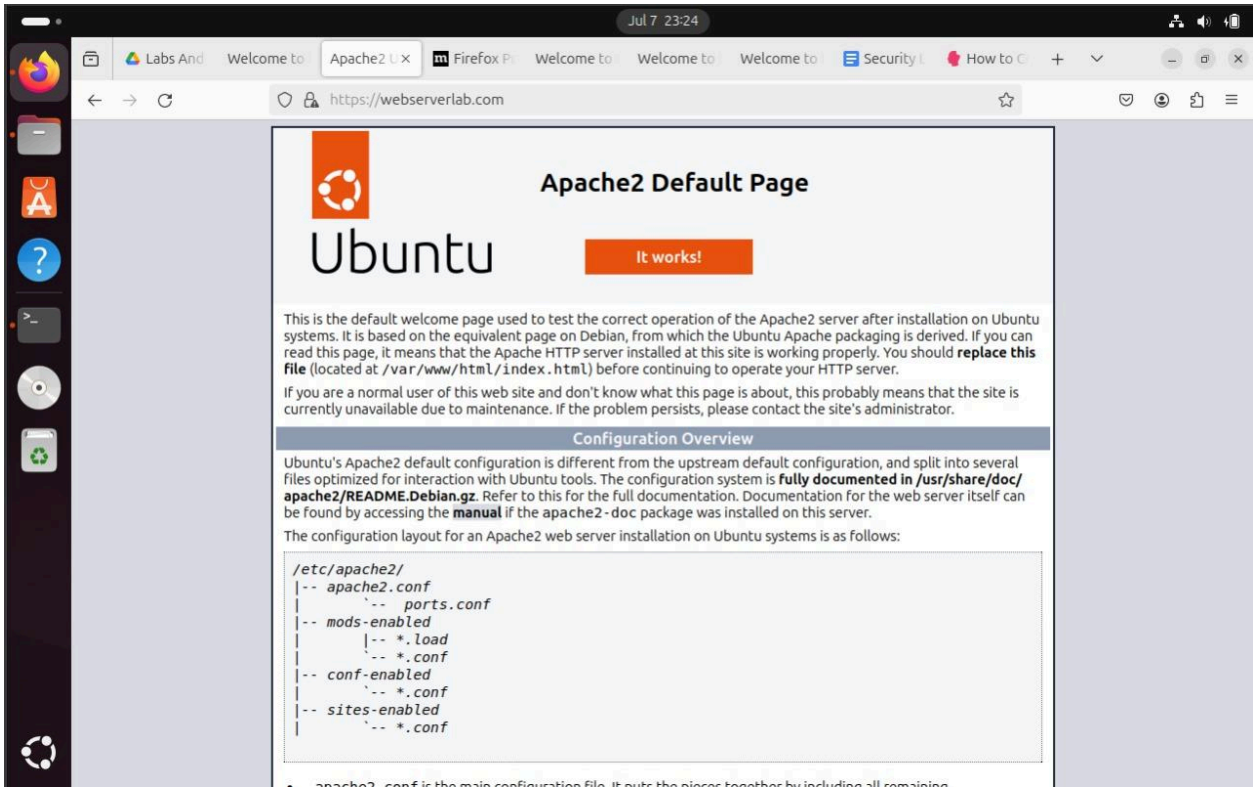
• Preference -> Advanced -> View Certificates

You will see a list of certificates that are already accepted by Firefox. From here, you can "import" our own certificate. Please import ca.crt, and select the following option: "Trust this CA to identify web sites". You will see that our CA's certificate is now in Firefox's list of the accepted certificates. Now, point the browser to https://example.com:4433.

Checkpoint – 1 (5 marks): Show this to your course teacher and explain what is happening. Since example.com points to 127.0.0.1, you can also use https://localhost:4433 to load a web page shown by the OpenSSL server. Please do so, describe and explain your observations.




Checkpoint – 2 (5 marks): Follow the same instructions for webserverlab.com and show this to your course teacher.



## Task-2: Deploy HTTPS into Apache

Now, you will deploy the HTTPS capability into Apache web server. At first, stop the Openssl webserver launched in the previous task. Now add the following lines into the example configuration file:

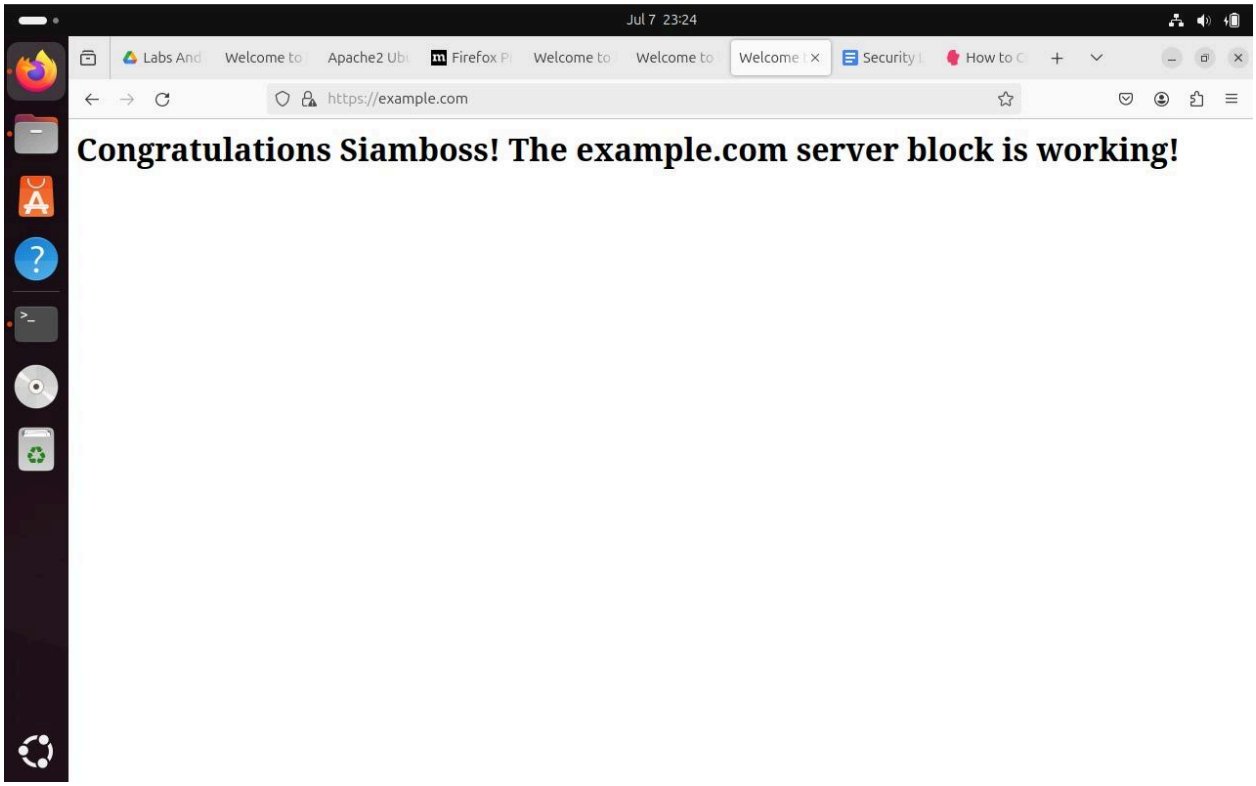
Apache is quite modular in the sense it supports the development of additional module which can add extended functionalities. For this lab, you will need to enable the ssl module in Apache which might not be enabled by default. Use the following command to enable the ssl module.

**sudo a2enmod ssl**

Next, use the following command to test the configuration.

**sudo apache2ctl configtest**

If a syntax is displayed onto the terminal, it indicates everything is okay.
Next restart the apache server using the restart command shown above.
Now, try to access the https://example.com. If everything is properly configured, you should be able to view the webpage in HTTPS.
If your browser is Firefox and it shows a warning, you can fix it by importing the CA certificate as described previously. If you use Chrome and it shows a similar warning, you can also import the CA certificate from the Manage certificate option under the Advanced setting in Chrome.
Checkpoint – 3 (5 marks): Access the https://example.com in your browser and show it to your teacher.




Checkpoint – 4 (5 marks): Set it up for webserverlab.com. Access it via HTTPS and show it to your teacher.