Apache Documentation Manual

05 BANANAL, Elmer Jr. Pasion

09 FERRER, Gavin Roy Llamido

23 AMBO, Melissa Denis

28 CUSTODIO, Danica Shiene Dela Masa

31 GONZALES, Darlene Joyce Buisan

33 LOPEZ, Kimberly Tangalin

37 PARIS, Lovelyn Degay

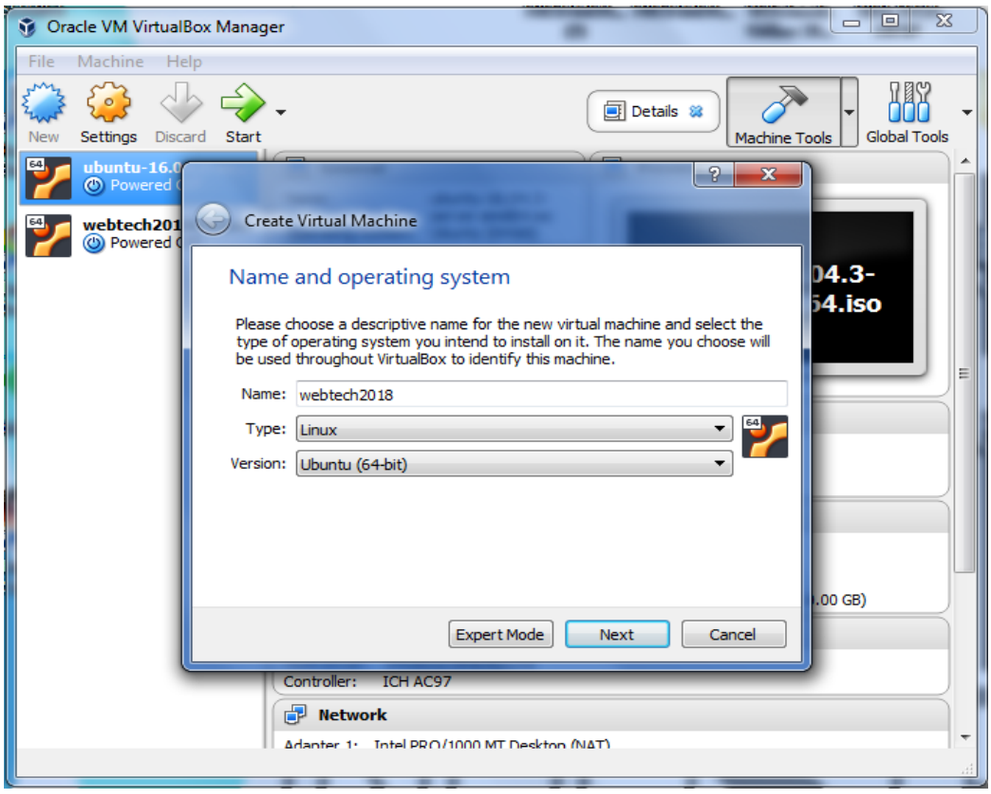
Table of Contents

I. Installation of Ubuntu Web Server

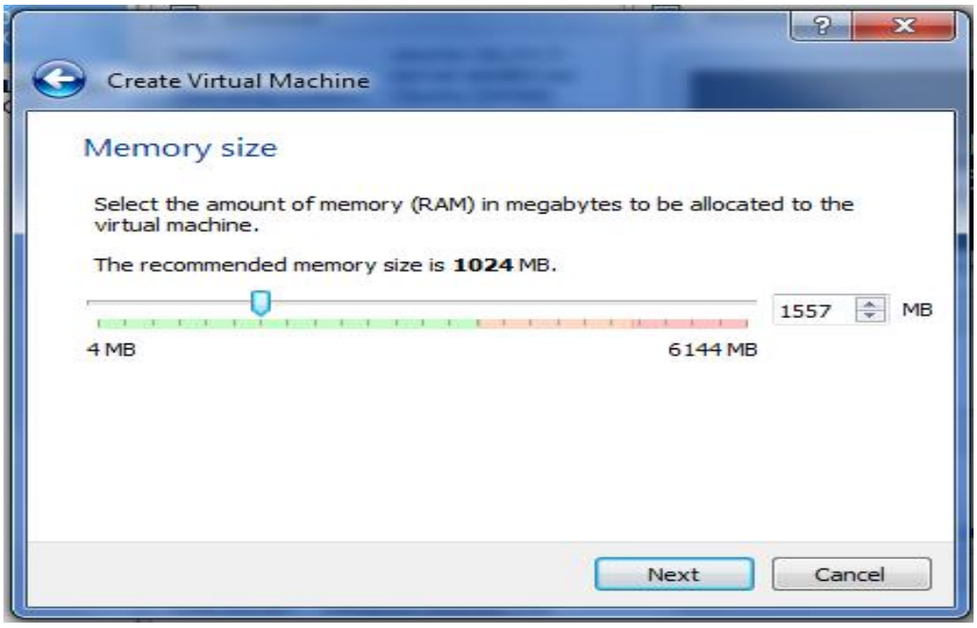
Requirements:

- Oracle VirtualBox 5.2.6
- Ubuntu Server 16.04 LTS Installer

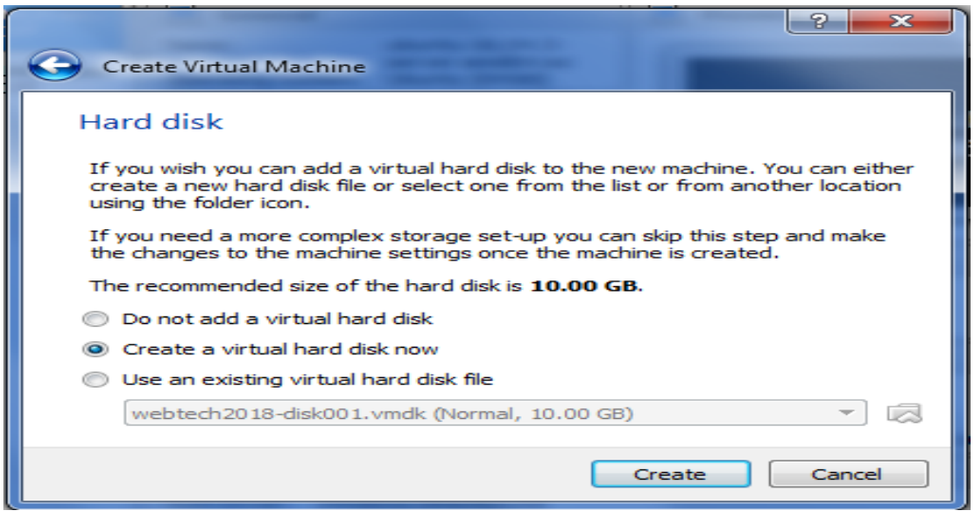
Install and launch Oracle VirtualBox 5.2.6. After installation, click "**New**" to add a virtual machine. Type the name of the new virtual machine.



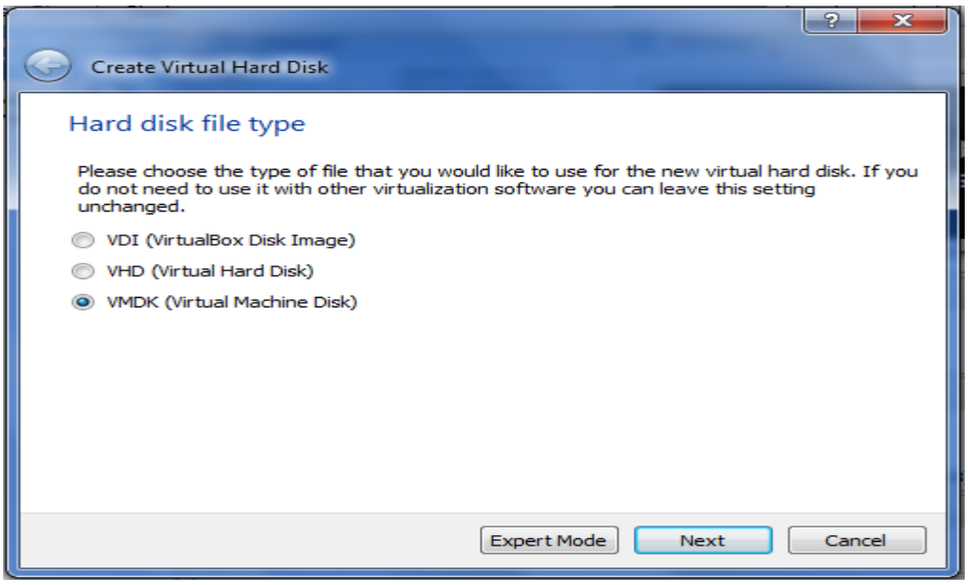
Select the amount of memory (RAM) in megabytes. It is recommended to be at 1024 MB but you may allocate more accordingly.



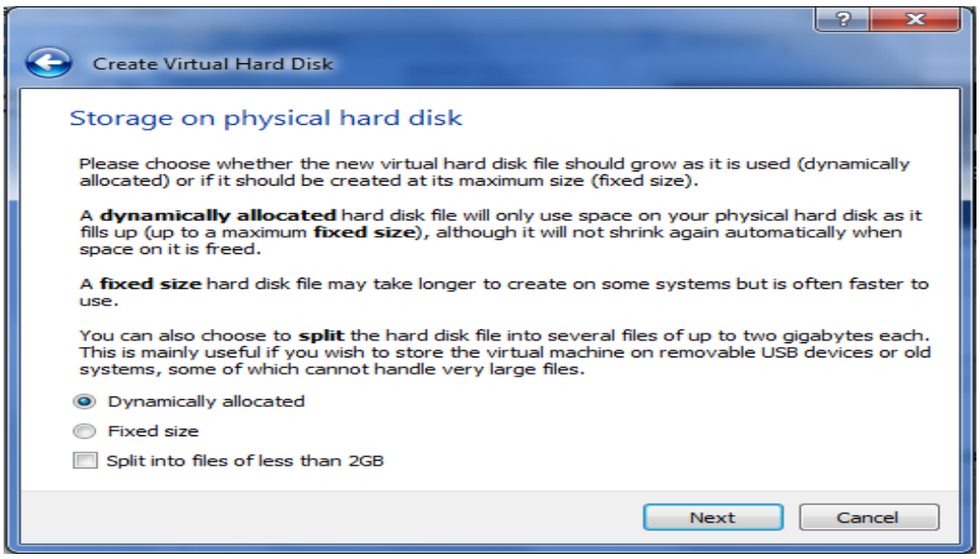
For the hard disk, proceed with the default which is ‘create a virtual hard disk’ then click “**Create**”.



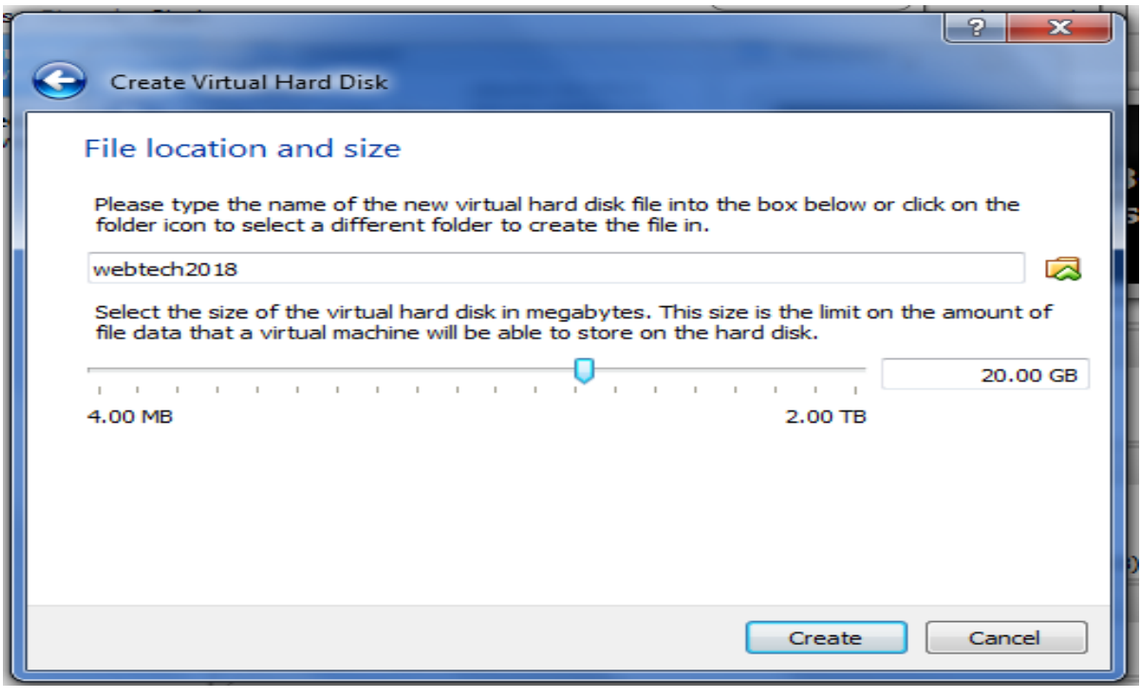
Choose the type of file that you would like to use for your new virtual hard disk. Click “**Next**”



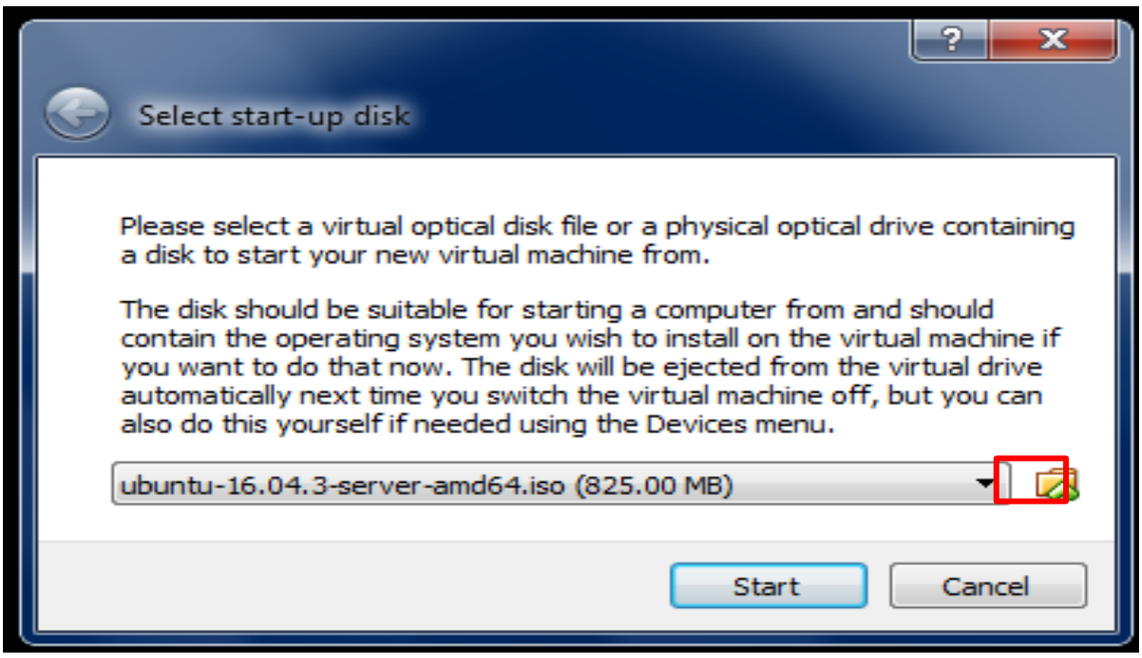
Choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum sixed (fixed size).



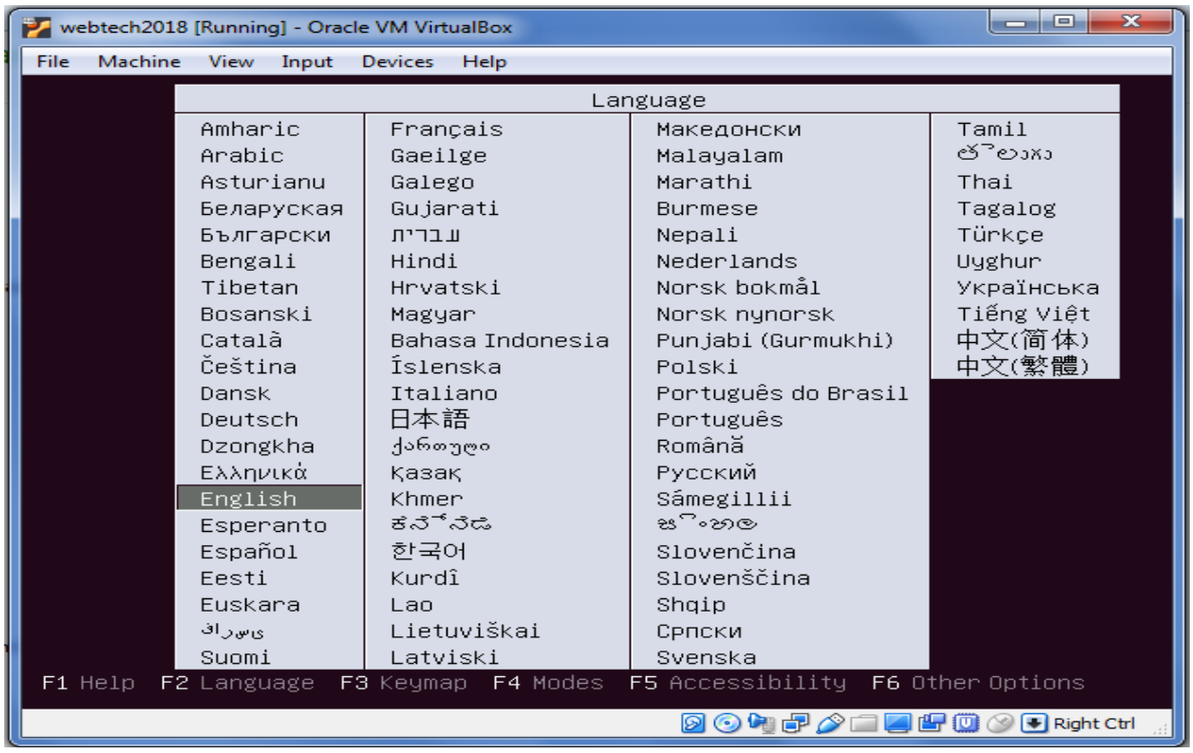
Next is the File location and size. Type the new name of the virtual disk file and select the size of the virtual hard disk.



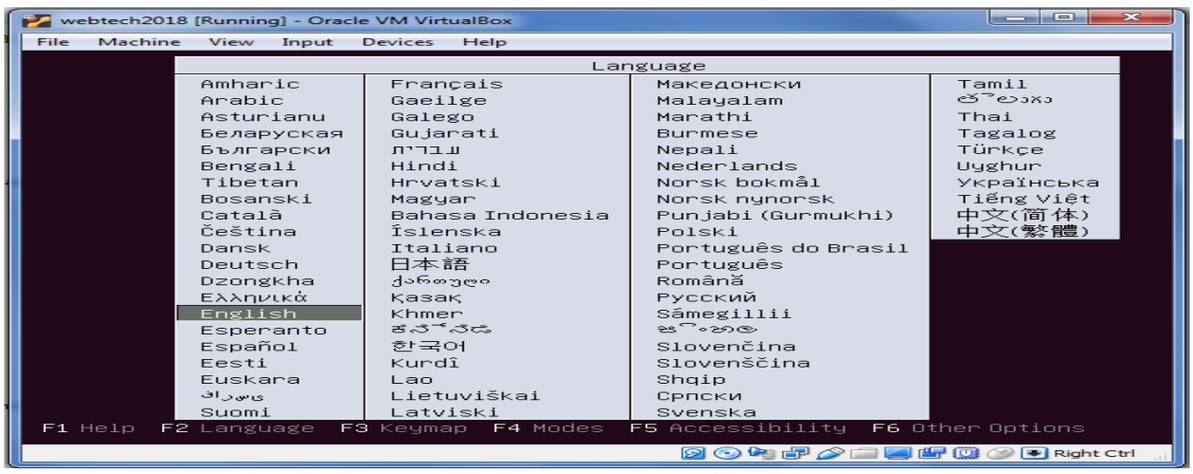
Then select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine form. Click "**Next**"



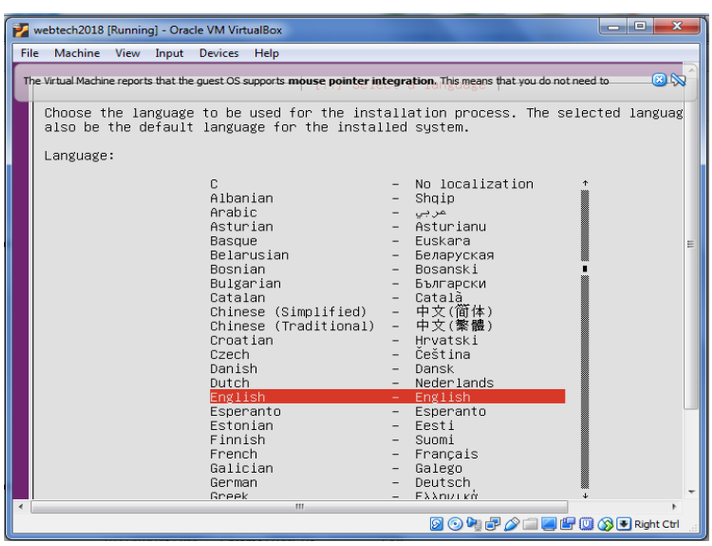
Choose your language.

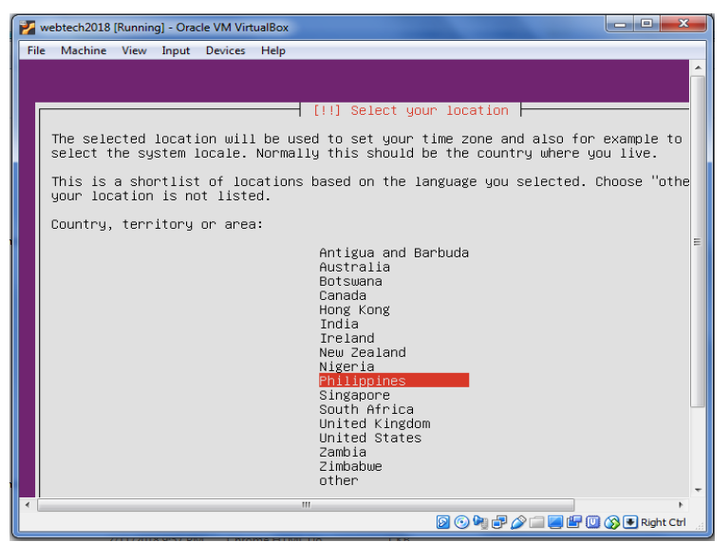


Then press **enter** on “**Install Ubuntu Server**.”

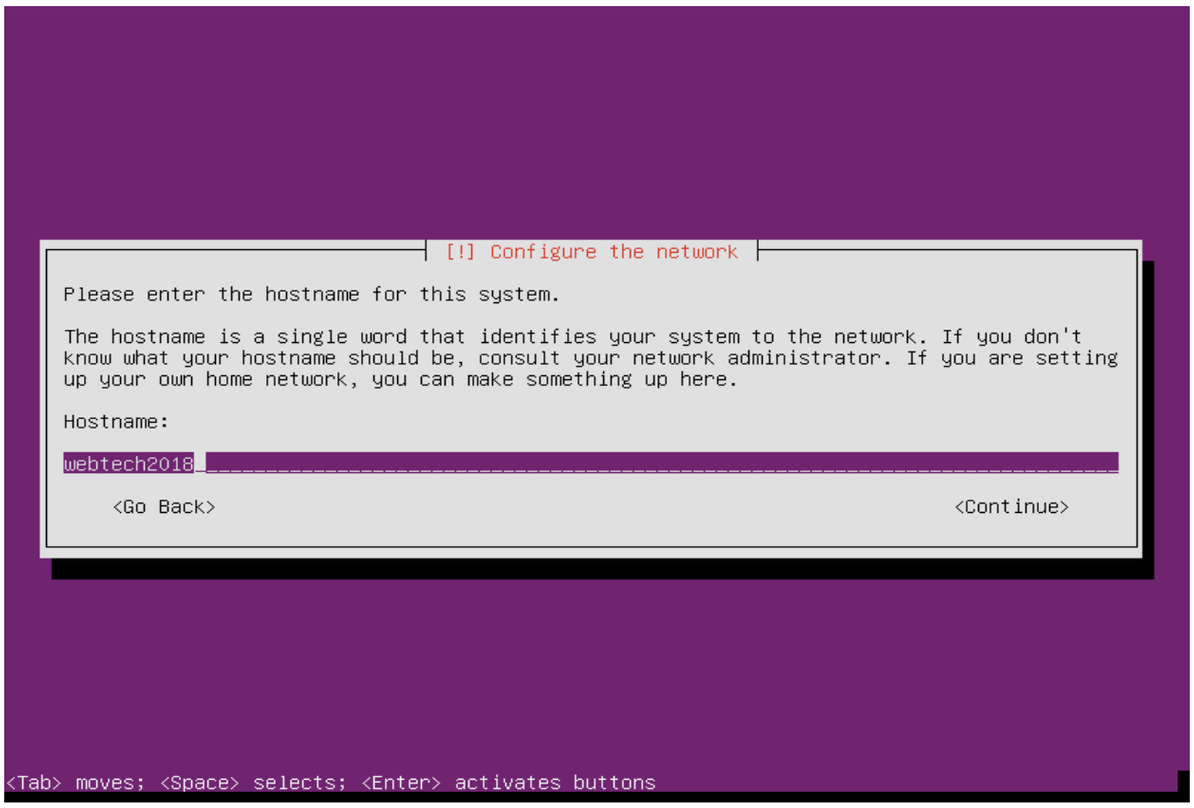


Select again the language to be used for the installation process and your country for the time zone.

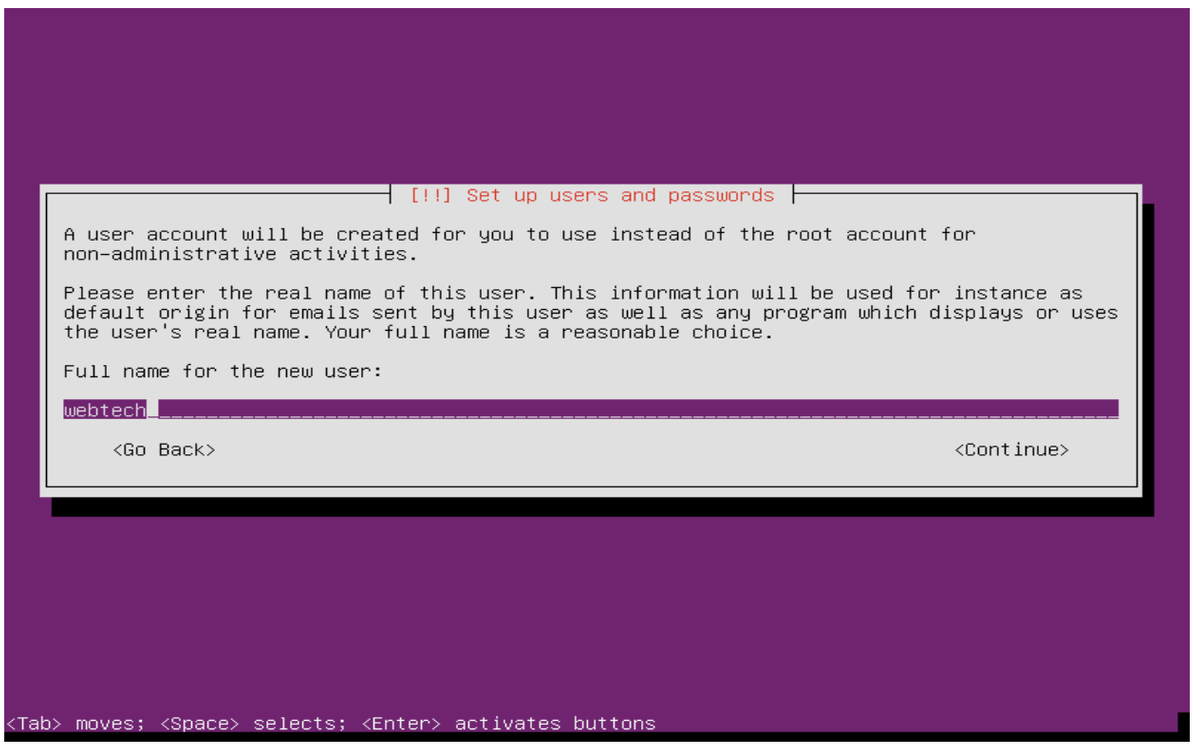




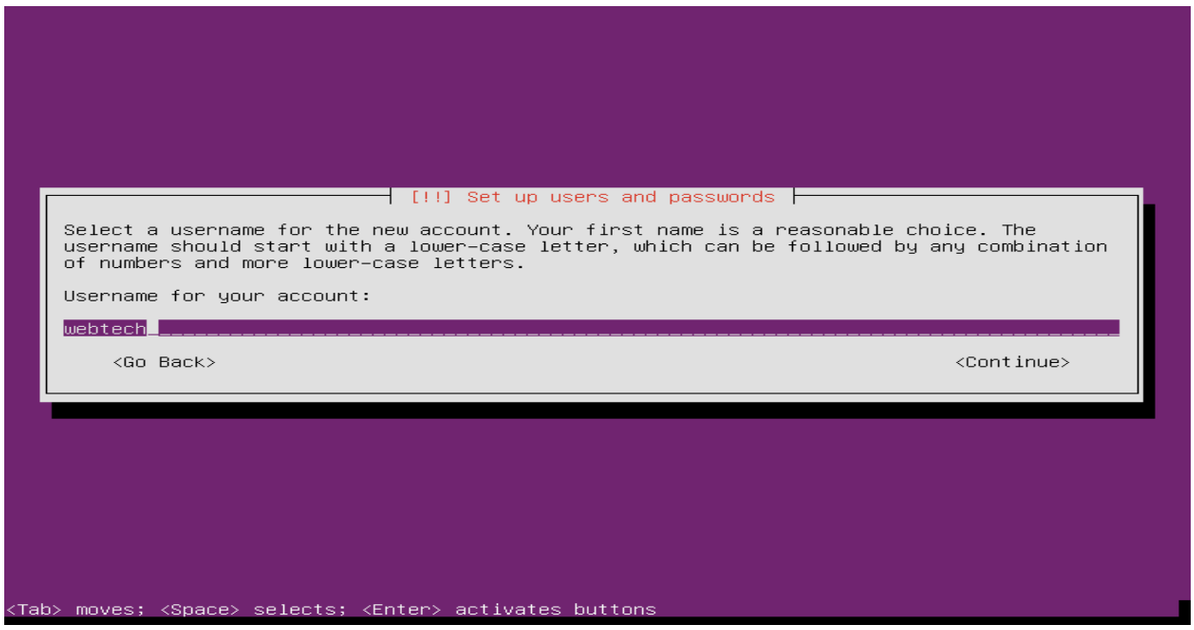
Enter your hostname, then continue.



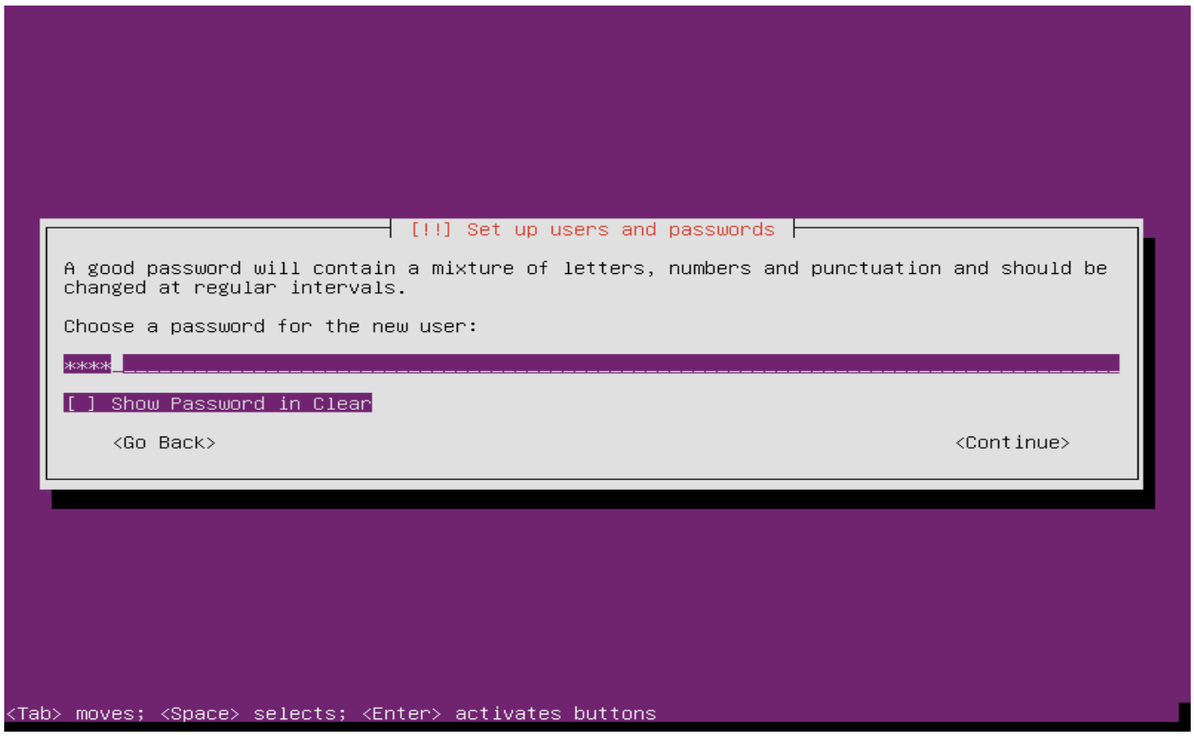
Enter the user name, then continue.



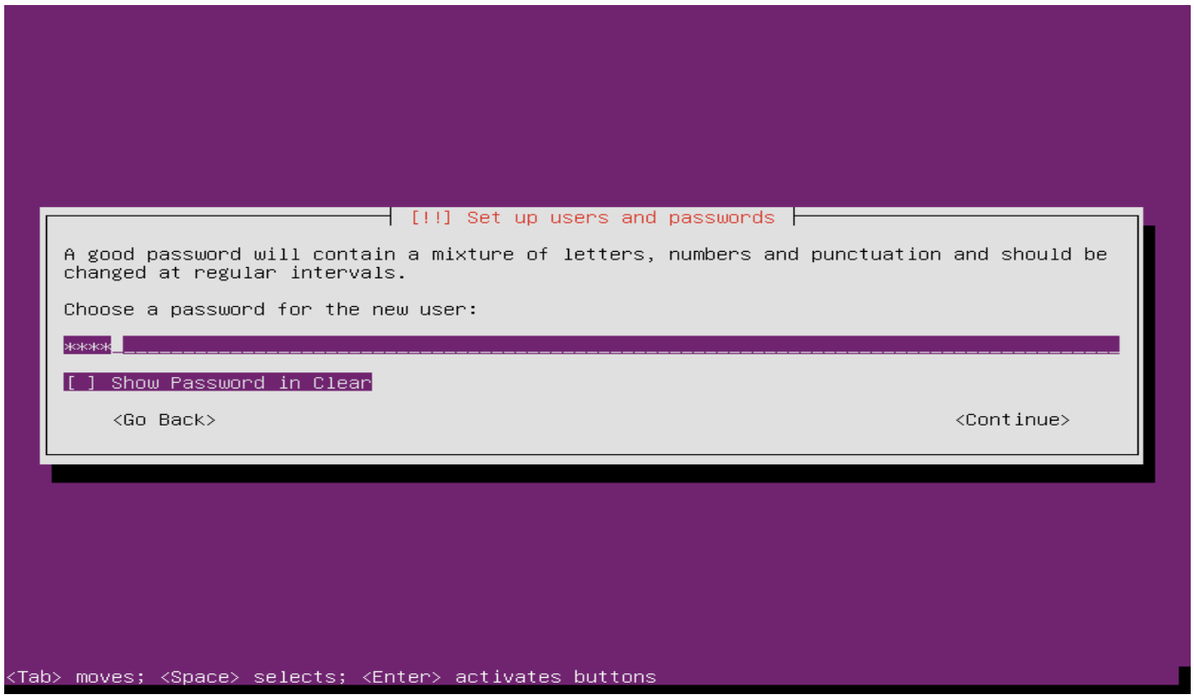
Create a username for your account and then continue.



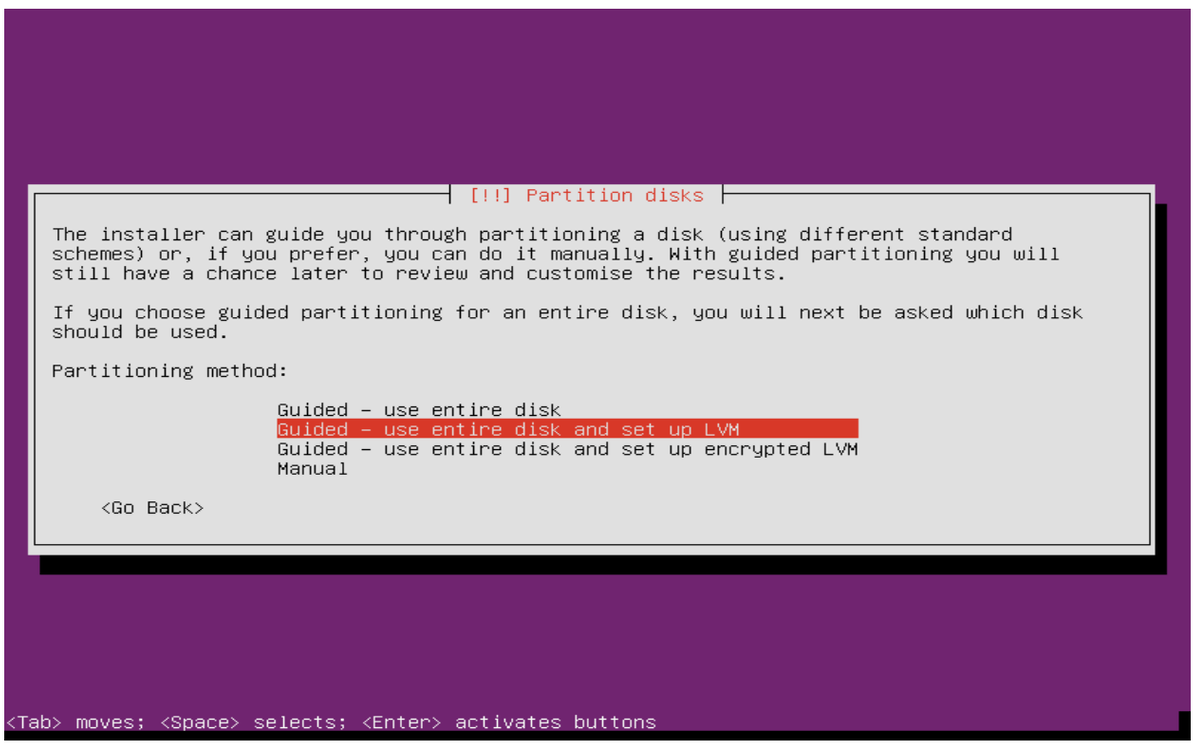
Then enter a password for the new user and click continue.



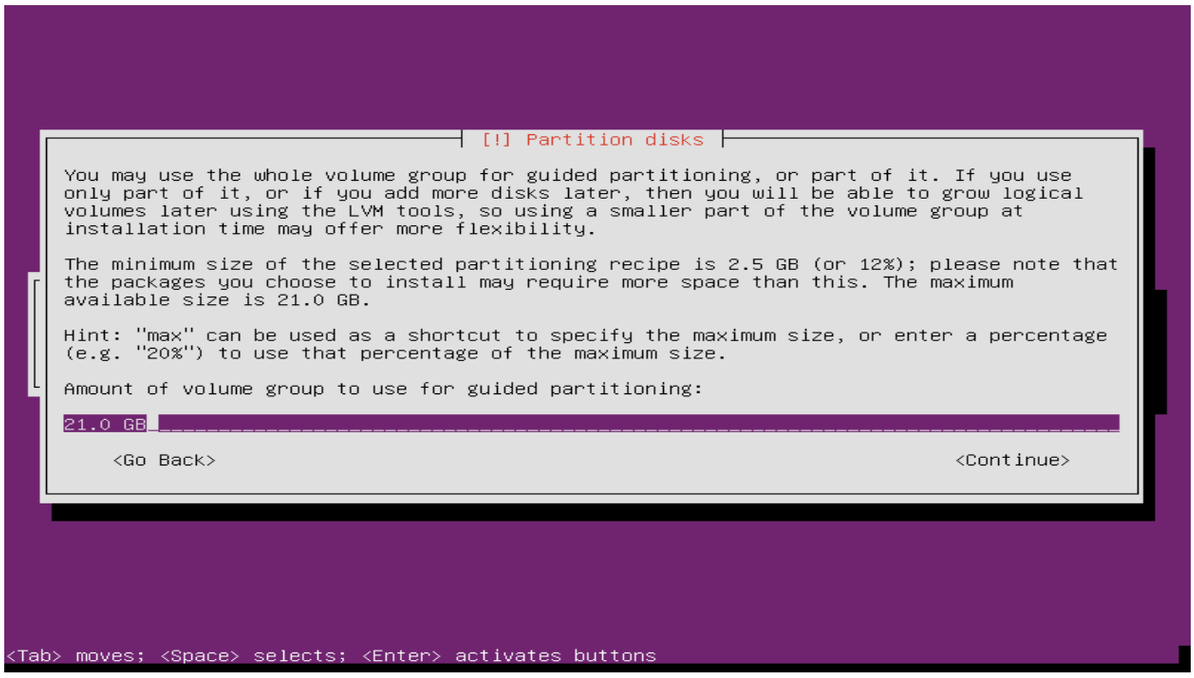
Then when it ask you to “**encrypt your home directory**” enter “**no**”.



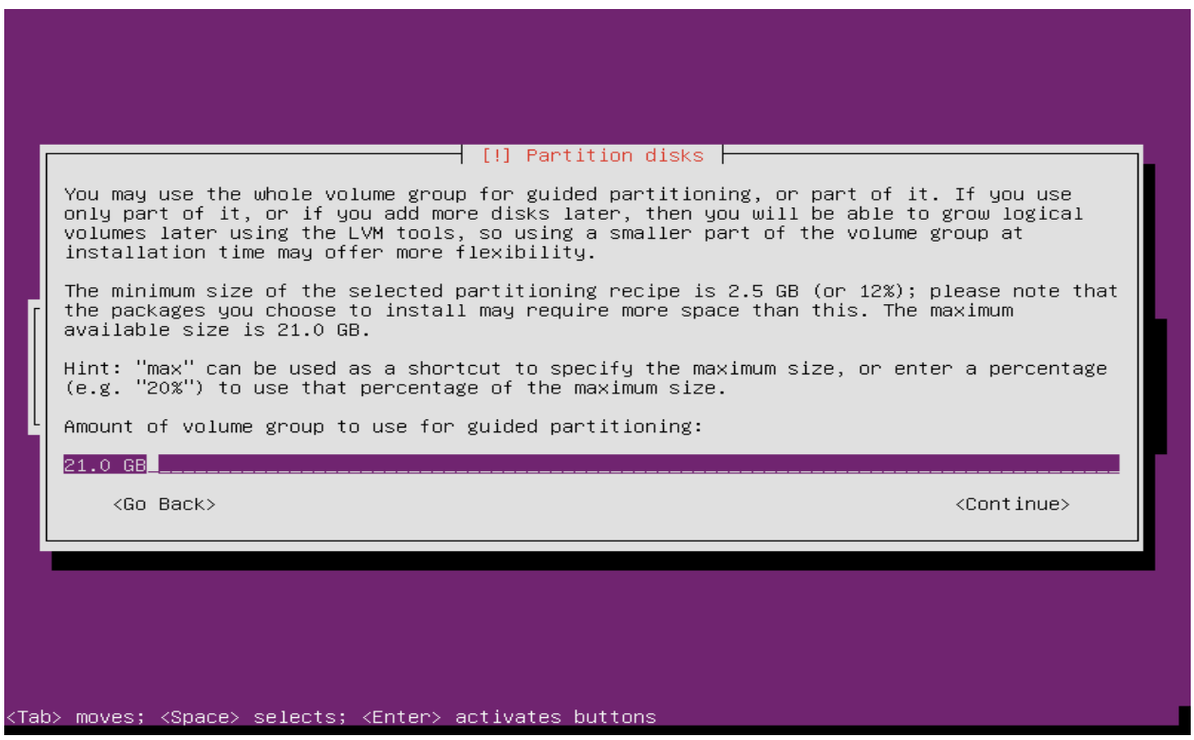
Then select a partitioning method. Select “**Guided – use entire disk and set up LVM**”.



Then select the amount of volume group to be used for the partitioning.In here, “**21.0 Gb**.” Then select continue.

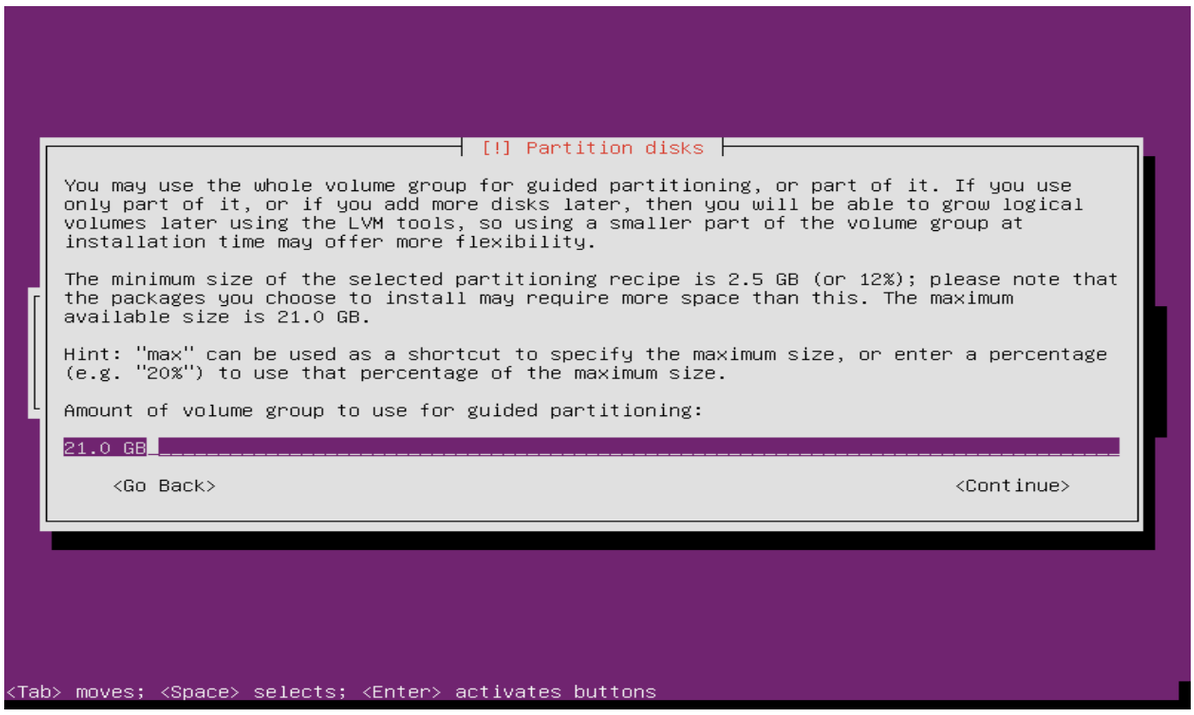


Then select “**yes**” to write the changes to disks.

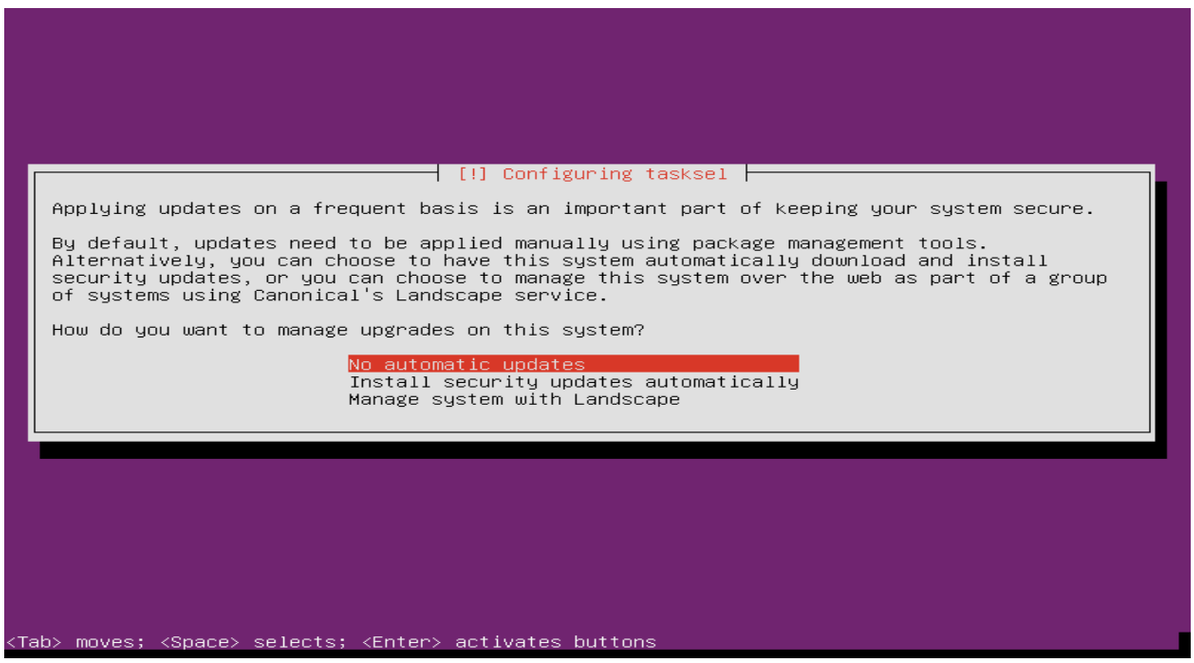


\

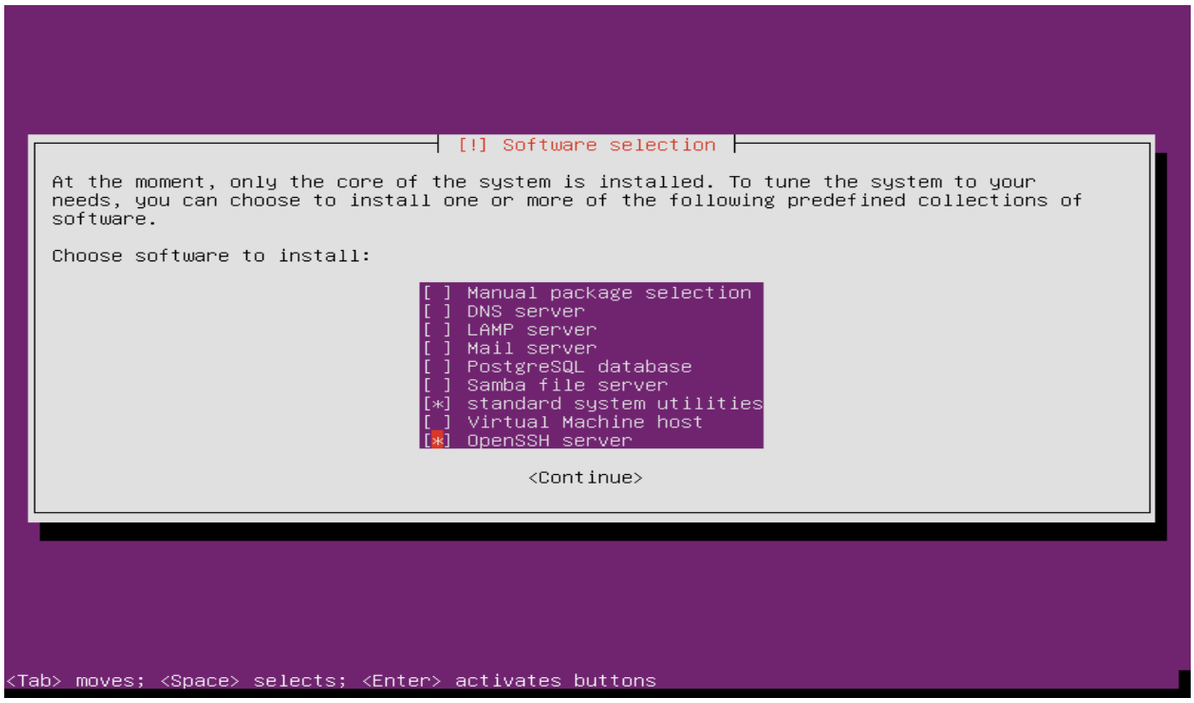
Wait as the system is being installed



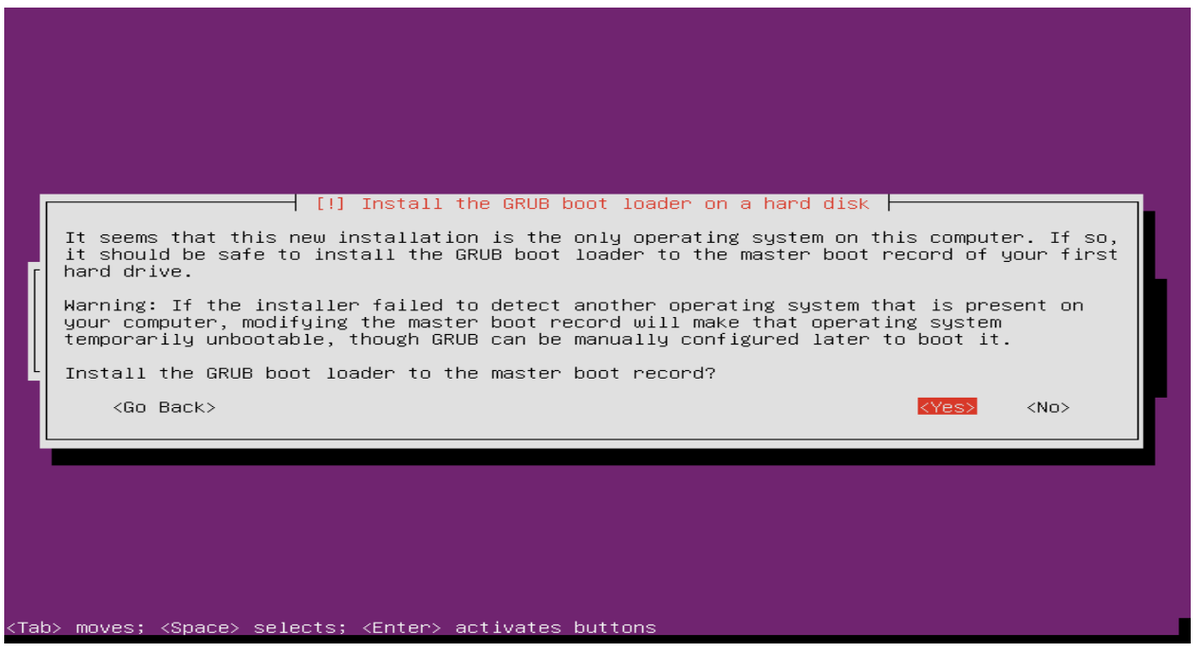
Manage your upgrades on your system. In this case, “**No automatic updates**”.



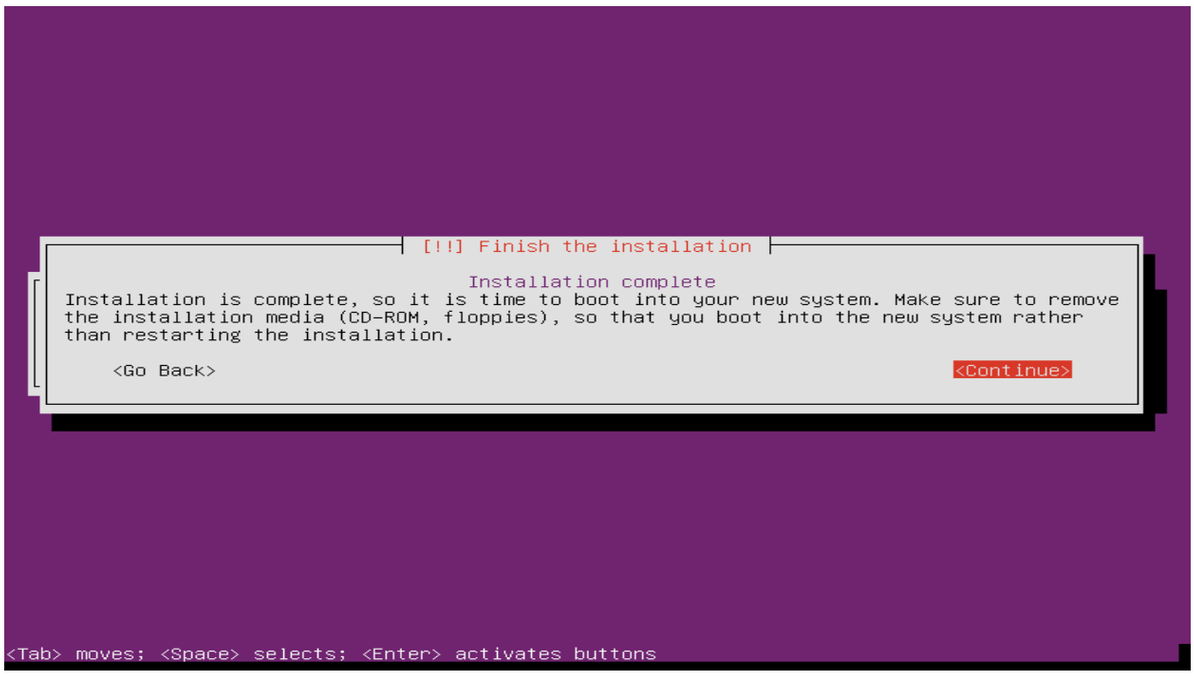
Choose the software to install, the standard system utilities is already chosen and for this installation, add **OpenSSH server** by using the arrow keys to move and the space bar to select. Click “**Continue**”.



Then install the **GRUB boot loader** on the hard disk by selecting “**yes**”.



Lastly, Finish the Installation and Select Continue.



II. Installation Apache

Type the following command to update Ubuntu.

```
webtech@webtech2018:~$ sudo apt-get update
```

To install apache2 in Ubuntu server, type:

```
webtech@webtech2018:~$ sudo apt-get install apache2
```

The apache is running in default when the server started:

```
webtech@webtech2018:~$ sudo service apache2 status
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Sun 2018-02-25 13:33:45 PHT; 8min ago
     Docs: man:systemd-sysv-generator(8)
  CGroup: /system.slice/apache2.service
           ├─1234 /usr/sbin/apache2 -k start
           ├─1237 /usr/sbin/apache2 -k start
           └─1238 /usr/sbin/apache2 -k start

Feb 25 13:33:41 webtech2018 systemd[1]: Starting LSB: Apache2 web server...
Feb 25 13:33:41 webtech2018 apache2[1210]:  * Starting Apache httpd web server apache2
Feb 25 13:33:45 webtech2018 apache2[1210]:  *
Feb 25 13:33:45 webtech2018 systemd[1]: Started LSB: Apache2 web server.
webtech@webtech2018:~$
```

III. Basic Commands

To manually stop the apache2, type:

```
webtech@webtech2018:~$ sudo service apache2 stop_
```

To manually start apache2, type:

```
webtech@webtech2018:~$ sudo service apache2 start
```

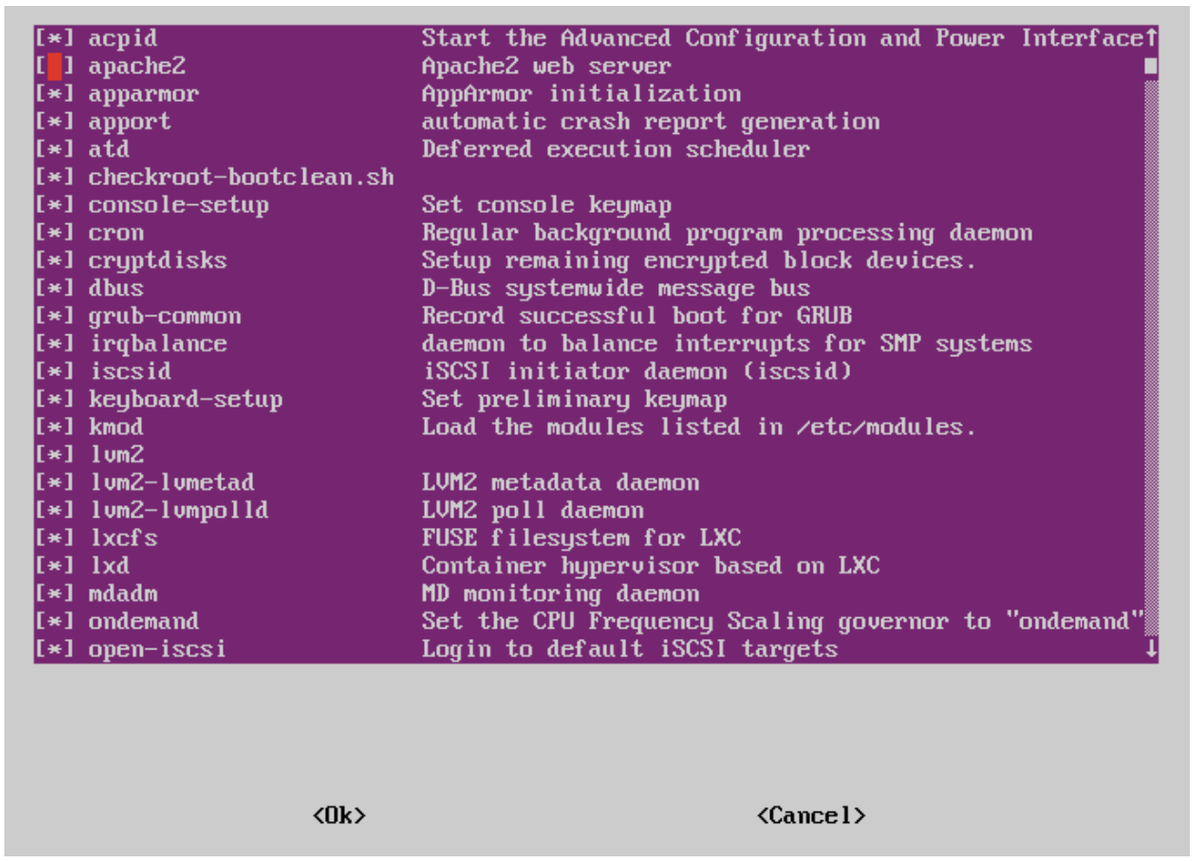
To manually restart apache2

```
webtech@webtech2018:/var/www/access$ sudo service apache2 restart
```

To disable apache2 from running when the server started, type:

```
webtech@webtech2018:~$ sudo apt-get intstall rcconf_
```

Next, uncheck apache2 from the list



Reboot the server to commit the changes made

```
webtech@webtech2018:~$ sudo reboot_
```

IV. Virtual Host

Virtual host refers to different websites running in a single web server. To create a virtual host, go to the root directory of Apache. This is where we will store all the resources for a website. The root directory of apache is as follows:

/var/www

We configured the server to contain the resources **(group4a, group4b, group4c)** to be used in creating a virtual host. The folder "**html**" contains the default welcome page of apache web server

```
webtech@webtech2018:/var/www$ ls
group4a  group4b  group4c  html
```

Next, we created a configuration file for the virtual host. Configuration files contains the settings as to how the virtual host would behave when accessed by a client. The configuration files are stored in the directory:

/etc/apache2/sites-available

Now inside the **sites-available** folder, we created the files named **group4a.conf, group4b.conf, and group4c.conf** as the configuration files for the three virtual host.

```
webtech@webtech2018:/etc/apache2/sites-available$ ls
000-default.conf  default-ssl.conf  group4a.conf  group4b.conf  group4c.conf
```

Using a text editor, we edited the configuration files. Remember to always use the command "**sudo**" so that the server would know that the command is from the administrator.

sudo nano group4a.conf

```
 GNU nano 2.5.3                File: group4a.conf

<VirtualHost *:80>
        ServerAdmin webmaster@group4a
        DocumentRoot /var/www/group4a
        ServerName www.group4a.org
        ServerAlias www.group4a.edu
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

<VirtualHost *:80> </VirtualHost> - all virtual host configuration are typed inside these tags.

***:80 –** the virtual host will be accessed using port 80.

ServerAdmin – the email of the site administrator.

DocumentRoot – the location for the resources of the website.

ServerName – the main domain name of the website.

ServerAlias – other domain names that could refer to the website.

After typing the configurations, save the file.

For debugging purposes, the following commands are used:

apachectl –S

```
webtech@webtech2018:/etc/apache2/sites-available$ apachectl -S
VirtualHost configuration:
*:80                   is a NameVirtualHost
         default server webtech2018.0.2.15 (/etc/apache2/sites-enabled/000-default.conf:1)
         port 80 namevhost webtech2018.0.2.15 (/etc/apache2/sites-enabled/000-default.conf:1)
         port 80 namevhost www.group4a.org (/etc/apache2/sites-enabled/group4a.conf:1)
                  alias www.group4a.edu
         port 80 namevhost www.group4b.org (/etc/apache2/sites-enabled/group4b.conf:1)
                  alias www.group4b.edu
         port 80 namevhost www.group4c.org (/etc/apache2/sites-enabled/group4c.conf:1)
                  alias www.group4c.edu
ServerRoot: "/etc/apache2"
Main DocumentRoot: "/var/www/html"
Main ErrorLog: "/var/log/apache2/error.log"
Mutex default: dir="/var/lock/apache2" mechanism=fcntl
Mutex watchdog-callback: using_defaults
PidFile: "/var/run/apache2/apache2.pid"
Define: DUMP_VHOSTS
Define: DUMP_RUN_CFG
User: name="www-data" id=33 not_used
Group: name="www-data" id=33 not_used
webtech@webtech2018:/etc/apache2/sites-available$
```

This command will show a description of the configuration file that may be used to uncover errors when debugging.

Next, we need to enable the virtual host configurations by typing the following command:

sudo a2ensite group4a.conf

```
webtech@webtech2018:/etc/apache2/sites-available$ sudo a2ensite group4a.conf
Enabling site group4a.
To activate the new configuration, you need to run:
  service apache2 reload
```

To commit the changes, restart apache by typing the following command

sudo service apache2 restart

In order for the client to access the websites, we need to modify the **hosts** file of the client to act as a DNS system. You can find the hosts file in the following directory:

Windows\System32\drivers\etc\host

```
22  192.168.254.103 www.group4a.org
23  192.168.254.103 www.group4b.org
24  192.168.254.103 www.group4c.org
```

Using a text editor, type in the **ipaddress** of the server and the **ServerName** in the host file. You can also use the **ServerAlias** as a domain name. By doing this, every request for www.group4a.org would direct the computer to the ip address of the server which would connect it to the virtual host.

Finally, in order to check if the virtual host is working, type the ServerName in any browser.

V. Content Compression

Compression is a function in apache that allows the server to compress files that will be access by the clients. This minimize the size of the resources, thus making clients access it easier and faster.

Using a text editor, open the configuration file of a virtual host and type the following commands inside the **<VirtualHost *:80> </VirtualHost>**

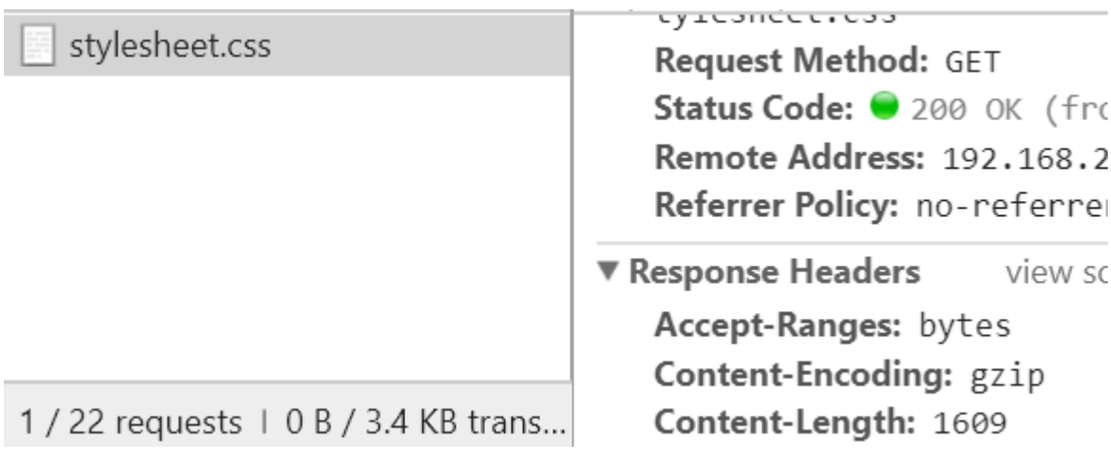
```
<VirtualHost *:80>
        ServerAdmin webmaster@group4a
        DocumentRoot /var/www/group4a
        ServerName www.group4a.org
        ServerAlias www.group4a.edu
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        <Directory /var/www/group4a>
                <IfModule mod_deflate.c>
                        <IfModule mod_filter.c>
                                AddOutputFilterByType DEFLATE text/html
                                AddOutputFilterByType DEFLATE text/css
                        </IfModule>
                </IfModule>
        </Directory>
</VirtualHost>
```

mod_filter.c is the configuration that is accessed for compression. You can add resource types by using "**AddOutputFilterByType DEFLATE type**". The command in the screenshot means that all html and css files will be compressed whenever the server serve it to a client.

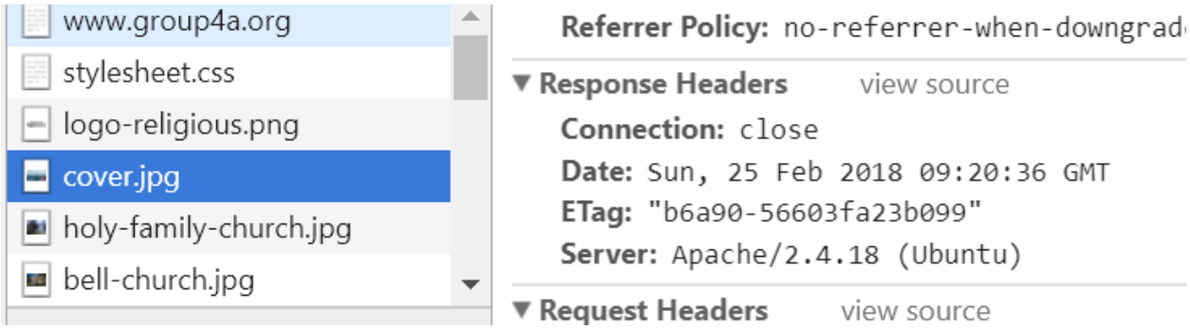
Restart apache to commit all configurations made

sudo service apache2 restart

To check whether the content compression is working, open your browser and open the **developer tools**. Click **Network**, reload the page and open the html file and the css file. As you can see from the screenshot below, the css file has a **content-encoding** that is equivalent to **gzip**. This means that the compression configuration is working.



Cover.jpg doesn't have a gzip because it was not included in the compression configuration.



VI. Content Caching

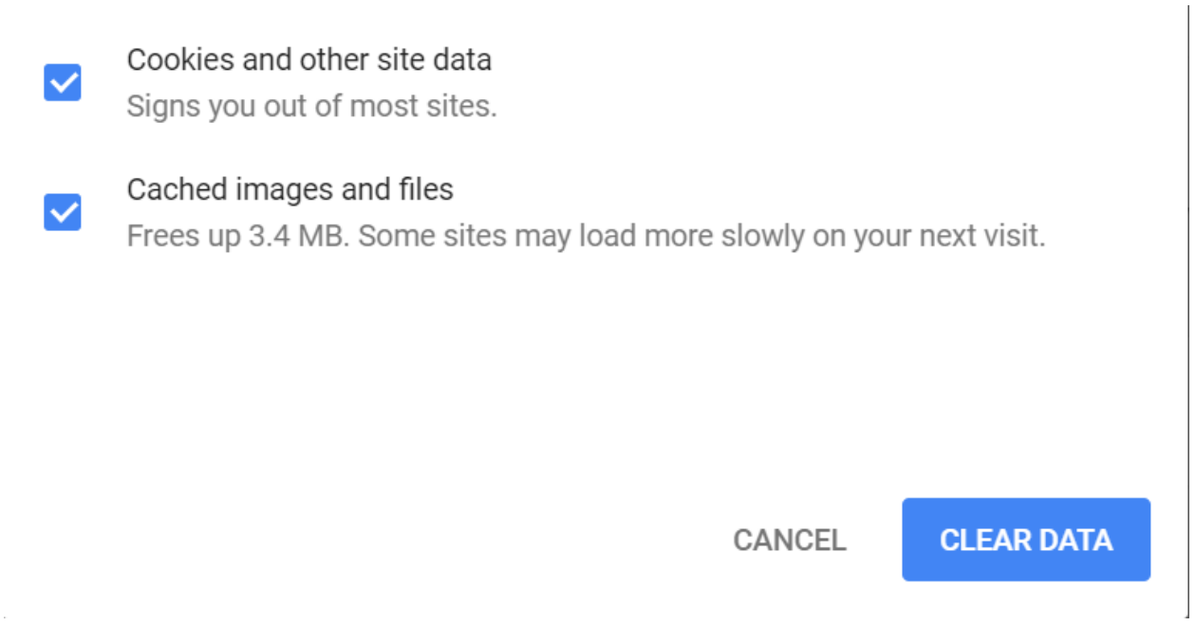
Apache's caching features allow the contents retrieved by the clients to be stored locally in the browser. This will speed up retrieval the next time the client access the resources.

In order to enable the caching, add the following settings to the configuration files.

```
<filesMatch "\.(png|jpg|gif)$">
Header set Cache-Control "max-age=86400, public"
</filesMatch>
```

This means that all files that has the extension **.png**, **.jpg**, and **.gif** will be cached and stored for **86400 seconds (24 hours).** After 24 hours, the cached content will be deleted.

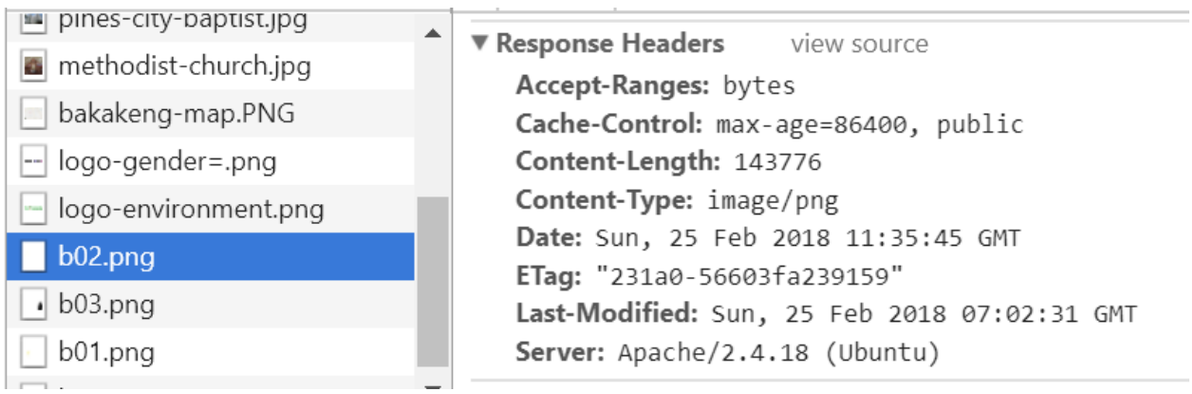
To check whether the resources are cached, go to your browser and clear all cached images and files.



Next load one of your website then close it afterwards. Then turn off your internet connection.

Try to load the website again, notice that the site can still load without internet connection.

Another way to check it is by using the **developer tool** of the browser. If the caching feature is enabled, the response header will contain the **max-age**.



VII. Content Negotiation

Content Negotiation is a feature of apache that can choose the best resource to be served base on the client's preferences.

A. Accept-Header

Let us first create a virtual host named **webtech1.negotiate.org**. A folder named **negotiate** is created to contain the resources for the virtual host. These resources are two files named **content.html and content.txt**

```
webtech@webtech2018:/var/www/negotiate$ ls
content.html  content.txt
```

Create a configuration file for the virtual host. Inside, put the basic settings like the ServerName, ServerAlias, etc. Next, add the configuration to enable **Multiviews**. Save and enable the configuration, then restart apache.

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName webtech1.negotiate.org
        ServerAlias webtech1.negotiate.edu
        DocumentRoot /var/www/negotiate
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        <Directory /var/www/negotiate>
                <IfModule mod_negotiation.c>
                        Options +Multiviews
                </IfModule>
        </Directory>
</VirtualHost>
```

Multiviews is one variant of content negotiation. It is enclosed in **mod_negotiation.c** for configuration. For example, the virtual host webtech1.negotiate.org contains two file with the same name but different types. One is an html, and the other is a plain txt file. If I accessed:

> **http://webtech.negotiate.org/content**

without typing the file extension, it's up to the server to negotiate base on the client's prefences.

```
Content-Location: content.html
Content-Type: text/html
```

According to the screenshot above, the server choose **content.html** because it best matches the client's preference.

B. Accept-Language

Inside the resource folder for webtech.negotiate.org, add the files **language.html.en** and **language.html.fil**

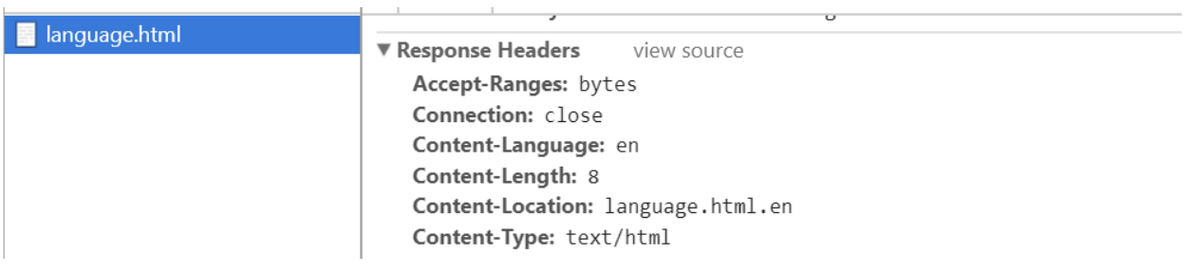
```
webtech@webtech2018:/var/www/negotiate$ ls
content.html  content.txt  language.html.en  language.html.fil
```

Add the following settings to the configuration file of **webtech.negotiate.org** virtual host

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName webtech1.negotiate.org
        ServerAlias webtech1.negotiate.edu
        DocumentRoot /var/www/negotiate
        AddLanguage fil .fil
        AddLanguage en .en
        LanguagePriority en fil
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        <Directory /var/www/negotiate>
                <IfModule mod_negotiation.c>
                        Options +Multiviews
                </IfModule>
        </Directory>
</VirtualHost>
```

AddLanguage – add a language and its extension

LanguagePriority – this tells the order of priority when serving the resources. In this case, en (English) has more priority than fil (Filipino) so English will be served first.



VIII. Access Control

Access Control is a feature of apache wherein the admin can configure and control who and what can access resources from a website.

First, let us create a virtual host named **webtech1.access.org** with folder named **access** that will contain the resource **index.html**

```
webtech@webtech2018:/var/www/access$ ls
index.html
```

Next open the configuration file for webtech1.access.org to edit the setting for the virtual host. Add the basic virtual host configurations (ServerName, ServerAlias, etc.) to the file.

A. Host and method Access Control

To control what host and method can access your website, add the following commands:

```
Require host .edu
Require host .org
Require method GET
```

Require host .edu – this means that all domain name that ends with *.edu can only access the website.

Require host .org – this means that all domain name that ends with *.org can only access the website.

Require method GET – this means that only the GET and HEAD method can access the website.

If the client trying to access the resources doesn't match to any access control configuration, the server would return a forbidden message.

B. Authorization and Authentication Access control

To configure the virtual host such that only clients that has user credentials can access the resources, let us create a file that will contain the users and passwords.

We created the folder **passwd** to contain the password configurations for the **webtech1.access.org** virtual host.

```
webtech@webtech2018:/etc/apache2/sites-available/passwd$ pwd
/etc/apache2/sites-available/passwd
```

Go inside the folder **passwd** and type the following command:

sudo htpasswd –c access 2161055

This command means that it will create a new password file using **htpasswd. "access"** is a file name for the password file (you may use other file name). **2161055** is the user.

It will prompt the administrator to type in the password for user xxx.

```
webtech@webtech2018:/etc/apache2/sites-available/passwd$ sudo htpasswd -c access
 2161055
New password:
Re-type new password:
Adding password for user 2161055
```

To add another user simply remove **–c.**

sudo htpasswd access newuser

Then add the following settings to a virtual host configuration file.

AuthName "Password"

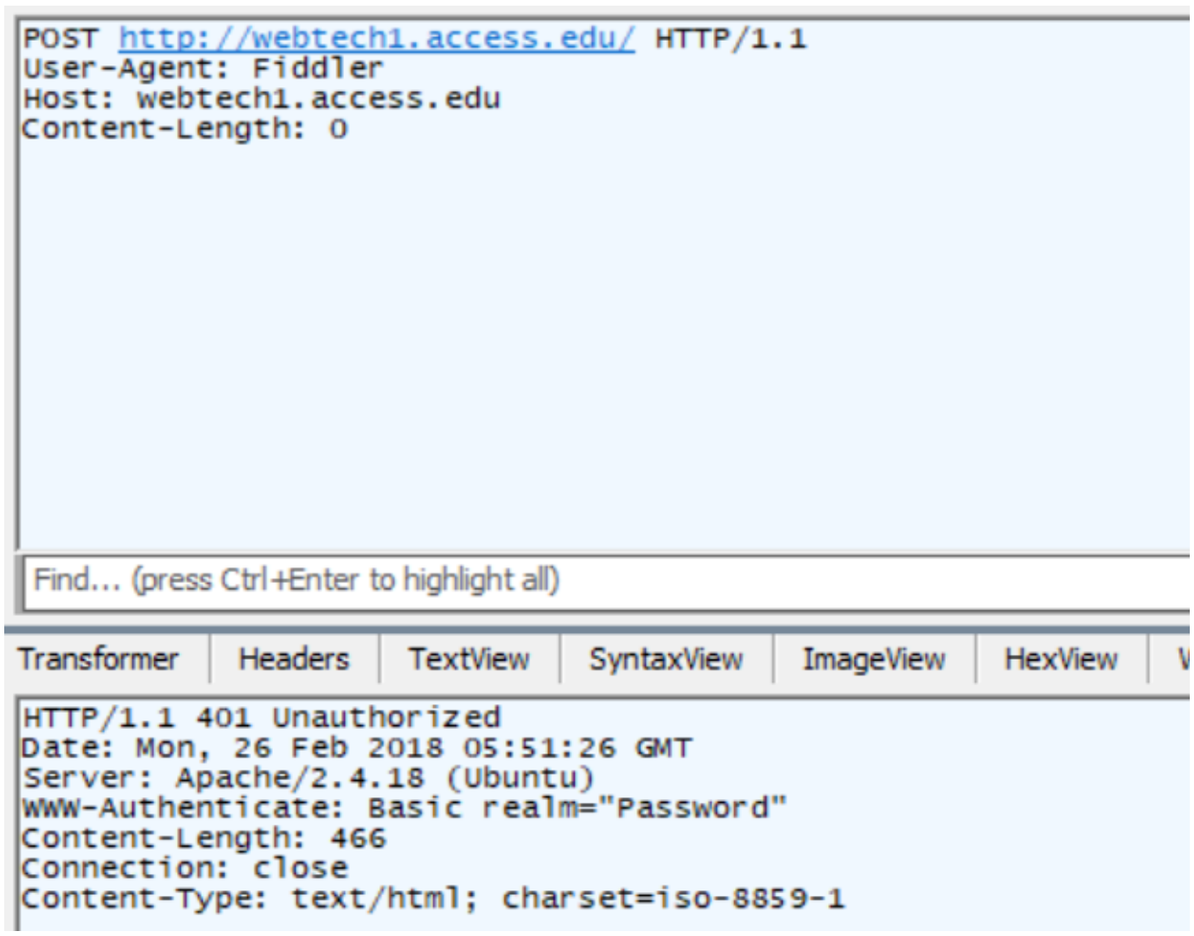
AuthName Basic

AuthUserFile passwordLocation

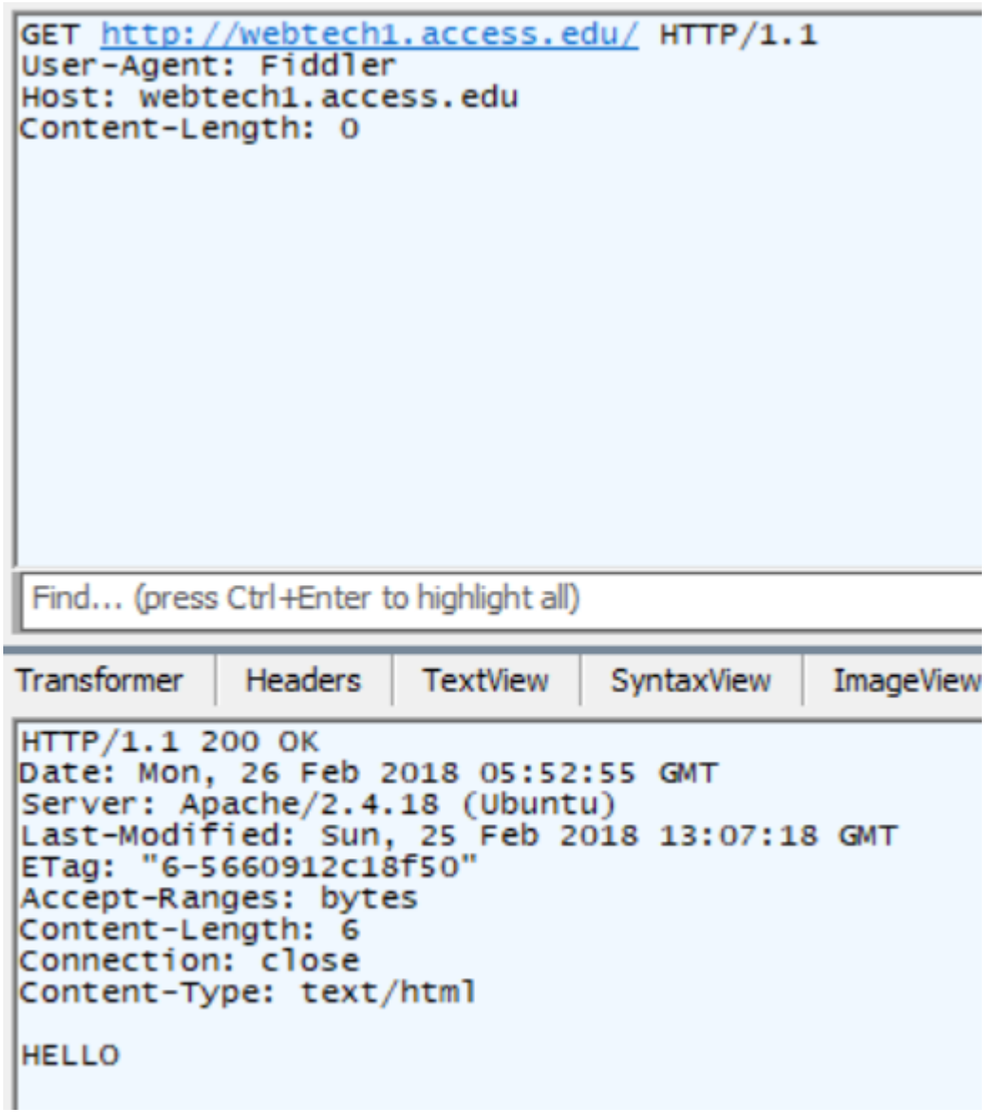
Require valid-user

```
<RequireAll>
        Require method POST
        AuthName "Password"
        AuthType Basic
        AuthUserFile /etc/apache2/sites-available/passwd/access
        Require valid-user
</RequireAll>
```

The configuration are enclosed in **<RequireAll> </RequireAll>.** This means that all commands inside the RequireAll tags must return true in order to be successful. The command above means that all clients who use the **POST** method must first be a valid-user.



The screenshot above shows that the post method requires authorization.



The screenshot above shows how the GET method has been given access.

IX. Server-side Includes

XBitHack Directive

Create new directory inside the document root. First, inside the new directory, create a plain text containing a simple format of a header text for the website page. See example below:

Next, along with the header.txt, create the index.html. The <!--#include virtual="header.txt" --> is the command that calls the header.txt inside the html file, and the <!--#echo var="DATE_LOCAL" --> calls the local time and date today.

```
 GNU nano 2.5.3                File: /var/www/ssi/index.html

<html>
   <body>
        <!--#include virtual="header.txt" -->
        <h1><!--#echo var="DATE_LOCAL" --></h1>
   </body>
</html>
```

Next, create and configure the SSI virtual host file for XBitHack directive.

$ sudo nano /etc/apache2/sites-available/ssi.conf

Inside the new blank host file, type the following line of codes:

```
 GNU nano 2.5.3          File: /etc/apache2/sites-available/ssi.conf

<VirtualHost *:80>
                ServerAdmin 2161842@slu.edu.ph
                ServerName webtech1.ssi.org
                DocumentRoot /var/www/ssi
                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

        <Directory /var/www/ssi >
                Options +Indexes +IncludesNOEXEC +FollowSymLinks
                AllowOverride all
                XBitHack On
        </Directory>
</VirtualHost>
```

Save and close the file when finished. (Ctrl + O > Enter > Ctrl + X)

Since XBitHack is used, it is required to add execution permission to the index.html that is embedded with another file using the command:

$ sudo chmod +x index.html

Next, enable the include module of Apache and the configuration of the SSL virtual host file (ssl.conf) using a2ensite. After the module and host are being enabled, it is required to reload or restart Apache after the configurations using:

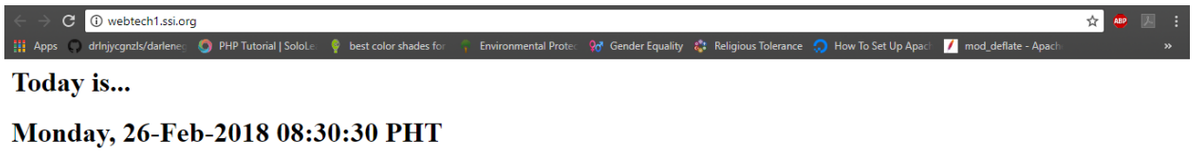
$ sudo a2enmod include

$ sudo a2ensite ssl.conf

$ sudo systemctl restart apache2

Note that $ sudo systemctl restart apache2 is only one of the options on how to reload Apache. You may use $ sudo service apache2 reload.

Test if the configuration is working. Open any browser and then type the domain name in the address bar. The output should be like this one:



Filename Extension-based

Follow the instructions stated above except for extension name of index. Instead of using .html as its extension file name, use .shtml. Also, another difference is the content of SSI virtual host file, instead of providing the line of codes shown above which is for XBitHack directive, provide this one:

```
 GNU nano 2.5.3          File: /etc/apache2/sites-available/ssi.conf

<VirtualHost *:80>
                ServerAdmin 2161842@slu.edu.ph
                ServerName webtech1.ssi.org
                DocumentRoot /var/www/ssi
                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined
                DirectoryIndex index.shtml
        <Directory /var/www/ssi >
                Options +Includes
                AllowOverride all
                AddType text/html .shtml
                AddOutputFilter INCLUDES .shtml
        </Directory>
</VirtualHost>
```

Actually, you'll get the same output as shown above.

X. SSL/TLS encryption

SSL or Secure Socket Layer, works using the combination of a public certificate and a private key. A SSL certificate is used to decrypt the content signed by an associated SSL key and being shared with any clients who are requesting the content. The SSL key is kept on the server, and encrypts the content sent to clients.

To create a self-signed SSL certificate, follow the instructions below:

Using the OpenSSL in Ubuntu Server, we can create a self-signed key and certificate using this single command:

$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt

After entering the command above, a series of questions is asked and is needed to fill out appropriately. The most important requested question here is the **Common Name**, which is the server's IP address or domain name. By this command, a **key file (path: /etc/ssl/private) and a certificate (path: /etc/ssl/certs)** are generated.

Create a configuration snippet to enable some features that will keep our server secure. This snippet will be created in the **/etc/apache2/conf-available** directory. A specified file name for this configurationis necessary like **ssl-params.conf**.

$ sudo nano /etc/apache2/conf-available/ssl-params.conf

This recommendation is by Remy van Elst on the Cipherli.st site to setup Apache SSL more secured. So, the file should contain this block of code shown below:

```
  GNU nano 2.5.3         File: /etc/apache2/conf-available/ssl-params.conf

# from https://cipherli.st/
# and https://raymii.org/s/tutorials/Strong_SSL_Security_On_Apache2.html

SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
SSLProtocol All -SSLv2 -SSLv3
SSLHonorCipherOrder On
# Disable preloading HSTS for now.  You can use the commented out header line that includes
# the "preload" directive if you understand the implications.
#Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains; preload"
Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains"
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff
# Requires Apache >= 2.4
SSLCompression off
SSLSessionTickets Off
SSLUseStapling on
SSLStaplingCache "shmcb:logs/stapling-cache(150000)"

SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.pem"
```

Save and close (Ctrl + O > Enter > Ctrl + X)

Next, create Apache SSL Virtual Host File. It is recommended to create a new host file for SSL and other host files instead of modifying the default host files. To create virtual host file, enter the command below:

$ sudo nano /etc/apache2/sites-available/ssl.conf

The '**ssl.conf'** on the command can be modified, it's your choice if what name of the host file do you want, but it's highly recommended that the name should be closely related to SSL.

Inside, insert the block of code shown below:

```
  GNU nano 2.5.3          File: /etc/apache2/sites-available/ssl.conf

<IfModule mod_ssl.c>
        <VirtualHost _default_:443>
                ServerAdmin 2161842@slu.edu.ph
                ServerName webtech1.secure.org
                DocumentRoot /var/www/ssl
                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

                SSLEngine on
                SSLCertificateFile      /etc/ssl/certs/apache-selfsigned.crt
                SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
        </VirtualHost>
</IfModule>
```

Save and close the file (Ctrl + O > Enter > Ctrl + X) when finished.

Create another virtual host file for SSL and configure the host such that the content pf the page is accessible via https or redirects the page as https request.

$ sudo nano /etc/apache2/sites-available/ssl.conf

This virtual host should contain the following block of code:

Save and close the file (Ctrl + O > Enter > Ctrl + X) when finished.

After the configuration of the two virtual hosts, we will now adjust the firewall of the Ubuntu server, the UFW.

First, install the UFW on Ubuntu Server using the command:

$ sudo apt-get install ufw

We need to adjust the settings of the firewall in order to allow for SSL traffic. After the installation, we can now enable the UFW using the command:

$ sudo ufw enable

To let in HTTPS traffic, we can now allow the ‘Apache Full’ profile using the command:

$ sudo ufw allow ‘Apache Full’

And to delete any redundant ‘Apache’ profile allowance, execute:

$ sudo ufw delete allow ‘Apache’

To check the firewall’s status together with the available applications and its action, enter:

$ sudo ufw status

Now that we already adjusted the firewall we can enable the SSL and header modules in Apache and configured SSL virtual hosts.

First, to enable SSL module and mod_headers, enter the commands:

$ sudo a2enmod ssl
$ sudo a2enmod headers

Now, enable the two SSL virtual hosts created a while ago with the a2ensite command:

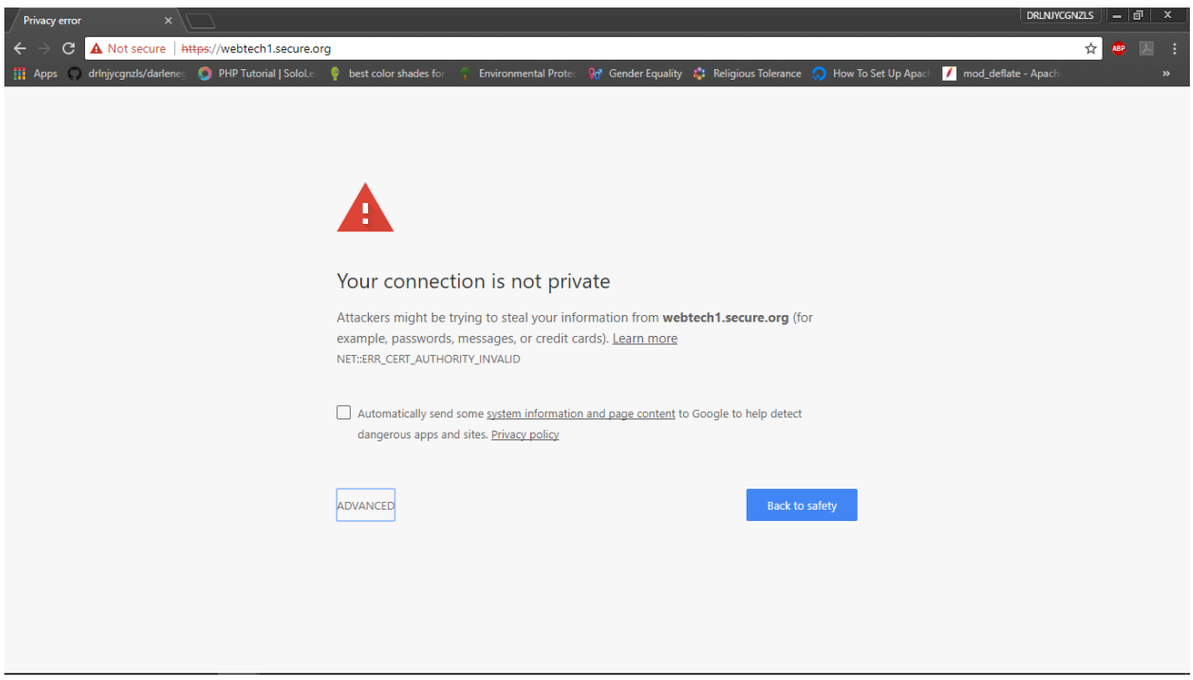
$ sudo a2ensite ssl.conf
$ sudo a2ensite ssl.vh.conf
Also, enable the ssl-params.conf
$ sudo a2ensite ssl-params.conf

At this point, the necessary modules and configured virtual hosts are enabled, we can check to make sure that there is an error free in our files by entering:

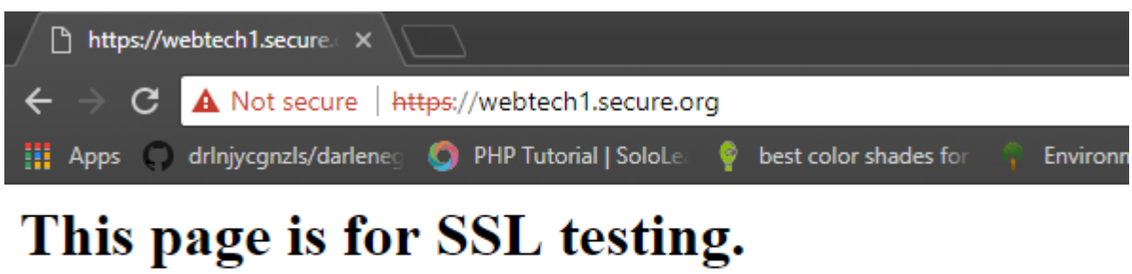
$ sudo apache2ctl configtest

Now, it's ready to test if SSL server is successfully working. Open any web browser in your host machine and type the domain name, for example webtech1.secure.org, into the address bar. Note: Make sure that you add the IP address and the domain name of the site in your host file **(path: C://windows/system32/drivers/etc/hosts**).

Your website page should show up like this one:



This is normal because the certificate that we created a while ago is not signed by one of your browser's trusted certificate authorities. Just click 'ADVANCED' and then to 'Proceed to webtech1.secure.org(unsafe)' to proceed to the website page you are expected. Notice that on the browser's address bar, there is an attention symbol over it, which means that the certificate cannot be validated and still encrypting the connection.



This page is for SSL testing.