**Chapter 2 :-WEB Server(Apache)**

**Introduction of Apache**

The Apache HTTP server is the most widely-used web server in the world. It provides many powerful features including dynamically loadable modules, robust media support, and extensive integration with other popular software.

* It is free to download and install.
* It is open source: the source code is visible to anyone and everyone, which basically enables anyone (who can rise up to the challenge) to adjust the code, optimize it, and fix errors and security holes. People can add new features and write new modules.
* It suits all needs: Apache can be used for small websites of one or two pages, or huge websites of hundreds and thousands of pages, serving millions of regular visitors each month. It can serve both static and dynamic content.

**Step 1: Install Apache**

Apache is available within Ubuntu's default software repositories, so we will install it using conventional package management tools.

We will begin by updating the local package index to reflect the latest upstream changes. Afterwards, we can install the apache2 package:

# sudo apt-get update

# sudo apt-get install apache2

After confirming the installation, apt-get will install Apache and all required dependencies.

**Step 2: Check your Web Server**

At the end of the installation process, Ubuntu 16.04 starts Apache. The web server should already be up and running.

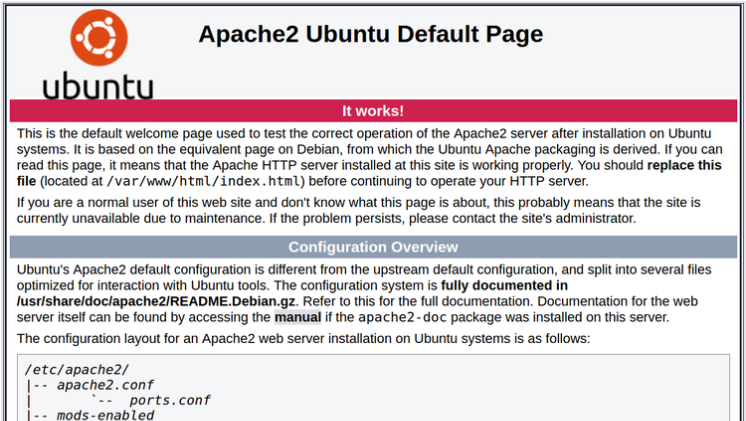
We can check with the systemd init system to make sure the service is running by typing:

# service apache2 restart

When you have your server's IP address or domain, enter it into your browser's address bar:

<http://server_domain_or_IP>

You should see the default Ubuntu 14.04 Apache web page, which should look something like this:



**Comparison between HTTP And HTTPS.**

**HTTP :**

Http stands for Hyper Text Transfer Protocol. It allows World Wide Web users to transferring information like image, text, video, music, graphic and other files on web pages. Http is basically used to access html pages and also other resources can be accessible using HTTP.

HTTP is a request-response protocol in the client-server computing model. When you enter http:// in front of the address tells the browser to connect over HTTP. For example, when you enter a URL (http://www.abc.com) in your web browser, this sends an HTTP command to the Web server to fetch and transfer the requested web page. Here, your web browser is your client and your website host as a server.

**Here is the fact of HTTP:**

* The Term HTTP is originated by Ted Nelson.
* HTTP connections uses a port 80 by default.
* HTTP URLs begin with “http://”.
* The first version of HTTP was introduced in 1991 that is HTTP V0.9.
* HTTP V1.0 is specified in RFC 1945 that officially introduced and recognized in 1996.
* HTTP V1.1 is specified in RFC 2616, and was released in January 1997.
* HTTP V2.0 is specified in RFC 7540 and was published in May 2015.

**HTTPS :**

HTTPS stands for Hypertext Transfer Protocol Secure. HTTPS is a protocol which uses an encrypted HTTP connection by transport-layer security.

Sometimes, the clients may be exchanging private information with a server, which needs to be secured for preventing some hacking issue. For this reason, HTTPS was developed by Netscape Corporation to allow authorization and secured transactions.

**Here is the fact of HTTP:**

* HTTPS uses a port 443 by default to transfer the information.
* HTTPS URLs begin with “https://”.
* The HTTPS is first used in HTTPS V1.1 and defined in RFC 2616.

**Generating SSL keys & Creating CA**

TLS, or transport layer security, and its predecessor SSL, secure sockets layer, are secure protocols created in order to place normal traffic in a protected, encrypted wrapper.

These protocols allow traffic to be sent safely between remote parties without the possibility of the traffic being intercepted and read by someone in the middle. They are also instrumental in validating the identity of domains and servers throughout the internet by establishing a server as trusted and genuine by a certificate authority.

**Step One — Activate the SSL Module**

SSL support actually comes standard in the Ubuntu 14.04 Apache package. We simply need to enable it to take advantage of SSL on our system.Enable the module by typing:

# sudo a2enmod ssl

After you have enabled SSL, you'll have to restart the web server for the change to be recognized:

# sudo service apache2 restart

With that, our web server is now able to handle SSL if we configure it to do so.

**Step Two — Create a Self-Signed SSL Certificate**

Let's start off by creating a subdirectory within Apache's configuration hierarchy to place the certificate files that we will be making:

# sudo mkdir /etc/apache2/ssl

Now that we have a location to place our key and certificate, we can create them both in one step by typing:

# sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out /etc/apache2/ssl/apache.crt

Let's go over exactly what this means.

**openssl:** This is the basic command line tool provided by OpenSSL to create and manage certificates, keys, signing requests, etc.

**req:** This specifies a subcommand for X.509 certificate signing request (CSR) management. X.509 is a public key infrastructure standard that SSL adheres to for its key and certificate managment. Since we are wanting to create a new X.509 certificate, this is what we want.

**-x509:** This option specifies that we want to make a self-signed certificate file instead of generating a certificate request.

**-nodes:** This option tells OpenSSL that we do not wish to secure our key file with a passphrase. Having a password protected key file would get in the way of Apache starting automatically as we would have to enter the password every time the service restarts.

**-days 365:** This specifies that the certificate we are creating will be valid for one year.

**-newkey rsa:2048:** This option will create the certificate request and a new private key at the same time. This is necessary since we didn't create a private key in advance. The rsa:2048 tells OpenSSL to generate an RSA key that is 2048 bits long.

**-keyout:** This parameter names the output file for the private key file that is being created.

**-out:** This option names the output file for the certificate that we are generating.

When you hit "ENTER", you will be asked a number of questions.

The most important item that is requested is the line that reads "Common Name (e.g. server FQDN or YOUR name)". You should enter the domain name you want to associate with the certificate, or the server's public IP address if you do not have a domain name.

The questions portion looks something like this:

Country Name (2 letter code) [AU]:US

State or Province Name (full name) [Some-State]:New York

Locality Name (eg, city) []:New York City

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Your Company

Organizational Unit Name (eg, section) []:Department of Kittens

Common Name (e.g. server FQDN or YOUR name) []:your\_domain.com

Email Address []:your\_email@domain.com

The key and certificate will be created and placed in your /etc/apache2/ssl directory.

**Step Three — Configure Apache to Use SSL**

Now that we have our certificate and key available, we can configure Apache to use these files in a virtual host file. You can learn more about how to set up Apache virtual hosts here.

Instead of basing our configuration file off of the 000-default.conf file in the sites-available subdirectory, we're going to base this configuration on the default-ssl.conf file that contains some default SSL configuration.

Open the file with root privileges now:

# sudo vim /etc/apache2/sites-available/default-ssl.conf

With the comments removed, the file looks something like this:

<IfModule mod\_ssl.c>

<VirtualHost \_default\_:443>

ServerAdmin webmaster@localhost

DocumentRoot /var/www/html

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

SSLEngine on

SSLCertificateFile /etc/ssl/certs/ssl-cert-snakeoil.pem

SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

<FilesMatch "\.(cgi|shtml|phtml|php)$">

SSLOptions +StdEnvVars

</FilesMatch>

<Directory /usr/lib/cgi-bin>

SSLOptions +StdEnvVars

</Directory>

BrowserMatch "MSIE [2-6]" \

nokeepalive ssl-unclean-shutdown \

downgrade-1.0 force-response-1.0

BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

</VirtualHost>

</IfModule>

This may look a bit complicated, but luckily, we don't need to worry about most of the options here.

We want to set the normal things we'd configure for a virtual host (ServerAdmin, ServerName, ServerAlias, DocumentRoot, etc.) as well as change the location where Apache looks for the SSL certificate and key.

In the end, it will look something like this. The entries in red were modified from the original file:

<IfModule mod\_ssl.c>

<VirtualHost \_default\_:443>

ServerAdmin admin@example.com

ServerName your\_domain.com

ServerAlias www.your\_domain.com

DocumentRoot /var/www/html

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

SSLEngine on

SSLCertificateFile /etc/apache2/ssl/apache.crt

SSLCertificateKeyFile /etc/apache2/ssl/apache.key

<FilesMatch "\.(cgi|shtml|phtml|php)$">

SSLOptions +StdEnvVars

</FilesMatch>

<Directory /usr/lib/cgi-bin>

SSLOptions +StdEnvVars

</Directory>

BrowserMatch "MSIE [2-6]" \

nokeepalive ssl-unclean-shutdown \

downgrade-1.0 force-response-1.0

BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

</VirtualHost>

</IfModule>

Save and exit the file when you are finished.

**Step Four — Activate the SSL Virtual Host**

Now that we have configured our SSL-enabled virtual host, we need to enable it.

We can do this by typing:

# sudo a2ensite default-ssl.conf

We then need to restart Apache to load our new virtual host file:

# sudo service apache2 restart

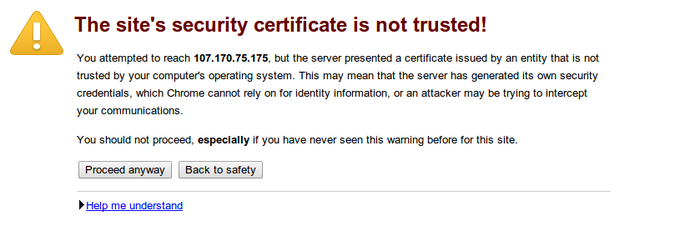
This should enable your new virtual host, which will serve encrypted content using the SSL certificate you created.

**Step Five — Test your Setup**

Now that you have everything prepared, you can test your configuration by visiting your server's domain name or public IP address after specifying the https:// protocol, like this:

<https://server_domain_name_or_IP>

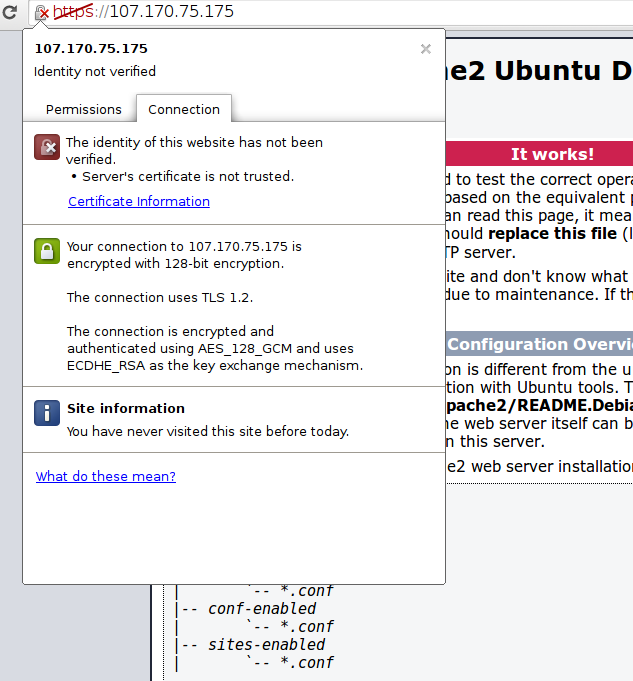
You will get a warning that your browser cannot verify the identity of your server because it has not been signed by one of the certificate authorities that it trusts.



This is expected since we have self-signed our certificate. While our certificate will not validate our server for our users because it has had no interaction with a trusted certificate authority, it will still be able to encrypt communication.

Since this is expected, you can hit the "Proceed anyway" button or whatever similar option you have in your browser.

You will now be taken to content in the DocumentRoot that you configured for your SSL virtual host. This time your traffic is encrypted. You can check this by clicking on the lock icon in the menu bar:



***Referance Url For Apache Virtual Host Congiguration :***

<https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-ubuntu-14-04-lts>