

What is Apache?

Apache is the most popular web server on the internet. It is used to serve more than half of all active websites.

Although there are many viable web servers that will serve your content, it is helpful to understand how Apache works because of its ubiquity.

In this article, we will examine some general configuration files and options that can be controlled within them. This article will follow the Ubuntu/Debian layout of Apache files, which is different from how other distributions build the configuration hierarchy.

The Apache File Hierarchy in Ubuntu and Debian

On Ubuntu and Debian, Apache keeps its main configuration files within the `/etc/apache2` folder:

```
cd /etc/apache2
ls -F
```

```
apache2.conf  envvars      magic        mods-enabled/  sites-available/
conf.d/       httpd.conf   mods-available/  ports.conf     sites-enabled/
```

There are a number of plain text files and some sub-directories in this directory. These are some of the more useful locations to be familiar with:

- **apache2.conf**: This is the main configuration file for the server. Almost all configuration can be done from within this file, although it is recommended to use separate, designated files for simplicity. This file will configure defaults and be the central point of access for the server to read configuration details.
- **ports.conf**: This file is used to specify the ports that virtual hosts should listen on. Be sure to check that this file is correct if you are configuring SSL.
- **conf.d/**: This directory is used for controlling specific aspects of the Apache configuration. For example, it is often used to define SSL configuration and default security choices.
- **sites-available/**: This directory contains all of the virtual host files that define different web sites. These will establish which content gets served for which requests. These are available configurations, not active configurations.
- **sites-enabled/**: This directory establishes which virtual host definitions are actually being used. Usually, this directory consists of symbolic links to files defined in the "sites-available" directory.
- **mods-[enabled,available]/**: These directories are similar in function to the sites directories, but they define modules that can be optionally loaded instead.

As you can see, Apache configuration does not take place in a single monolithic file, but instead happens through a modular design where new files can be added and modified as needed.

Looking at the Apache2.conf File

The main configuration details for your Apache server are held in the `/etc/apache2/apache2.conf` file.

This file is divided into three main sections: configuration for the global Apache server process, configuration for the default server, and configuration of Virtual Hosts.

In Ubuntu and Debian, the majority of the file is for global definitions, and the configuration of the default server and virtual hosts is handled at the end, by using the `"Include ..."` directive.

The `"Include"` directive allows Apache to read other configuration files into the current file at the location that the statement appears. The result is that Apache dynamically generates an overarching configuration file on startup.

If you scroll to the bottom of the file, there are a number of different `"Include"` statements. These load module definitions, the `ports.conf` document, the specific configuration files in the `"conf.d/"` directory, and finally, the Virtual Host definitions in the `"sites-enabled/"` directory.

We will focus on the first part of the file to learn how Apache defines its global settings.

Global Configuration Section

This section is used to configure some options that control how Apache works as a whole.

There are some interesting options you may want to modify in this section:

Timeout

By default, this parameter is set to `"300"`, which means that the server has a maximum of 300 seconds to fulfill each request.

This is probably too high for most set ups and can safely be dropped to something between 30 and 60 seconds.

KeepAlive

This option, if set to `"On"`, will allow each connection to remain open to handle multiple requests from the same client.

If this is set to `"Off"`, each request will have to establish a new connection, which can result in significant overhead depending on your setup and traffic situation.

MaxKeepAliveRequests

This controls how many separate request each connection will handle before dying. Keeping this number high will allow Apache to serve content to each client more effectively.

Setting this value to 0 will allow Apache to serve an unlimited amount of request for each connection.

KeepAliveTimeout

This setting specifies how long to wait for the next request after finishing the last one. If the timeout threshold is reached, then the connection will die.

This just means that the next time content is requested, the server will establish a new connection to handle the request for the content that make up the page the client is visiting.

Install Apache Webserver

Prior to install apache server, let us update our Ubuntu server:

To do that, run:

```
sudo apt-get update
```

Now, install apache web server using the following command:

```
sudo apt-get install apache2
```

After installing apache server, let us test whether the webserver is working properly or not by navigating to the URL **http://ip-address/**.



Apache2 Ubuntu Default Page

ubuntu

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective `*-available/` counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf`. See their respective man pages for detailed information.
- The binary is called `apache2`. Due to the use of environment variables, in the default configuration, `apache2` needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Document Roots

By default, Ubuntu does not allow access through the web browser to *any* file apart of those located in `/var/www`, **public_html** directories (when enabled) and `/usr/share` (for web applications). If your site is using a web document root located elsewhere (such as in `/srv`) you may need to whitelist your document root directory in `/etc/apache2/apache2.conf`.

The default Ubuntu document root is `/var/www/html`. You can make your own virtual hosts under `/var/www`. This is different to previous releases which provides better security out of the box.

Reporting Problems

Please use the `ubuntu-bug` tool to report bugs in the Apache2 package with Ubuntu. However, check **existing bug reports** before reporting a new bug.

Please report bugs specific to modules (such as PHP and others) to respective packages, not to the web server itself.

As you see in the above picture, apache webserver is working.

Now, let us proceed to setup virtual hosts in Apache web server.

Setup Apache Virtual Hosts

1. Create Virtual Directories

Now, let us proceed to setup virtual hosts. As I mentioned earlier, I am going to host two virtual hosts called “**unixmen1.local**”, and “**unixmen2.local**”.

Create a public directory to place the two virtual hosts data.

First, let us create a directory for unixmen1.local site:

```
sudo mkdir -p /var/www/html/unixmen1.local/public_html
```

Then, create the directory for unixmen2.local site:

```
sudo mkdir -p /var/www/html/unixmen2.local/public_html
```

2. Setting Up Ownership and Permissions

The above directories are owned by root user now. We should change the ownership of these two directories to the regular user.

```
sudo chown -R $USER:$USER /var/www/html/unixmen1.local/public_html/
```

```
sudo chown -R $USER:$USER /var/www/html/unixmen2.local/public_html/
```

The “**\$USER**” variable indicates the currently logged in user.

Set the read permissions to the Apache web root (/var/www/html/) directory, so that everyone can read files from that directory.

```
sudo chmod -R 755 /var/www/html/
```

We have created the directories for holding the websites data and assigned the necessary permissions and ownership to them.

4. Create Sample pages for Virtual Hosts

Now, we have to create the sample pages to be served through the websites.

First, let us create a sample page to the **unixmen1.local** virtual host.

Create a ‘index.html’ for unixmen1.local virtual host,

```
sudo vi /var/www/html/unixmen1.local/public_html/index.html
```

Add the following contents:

```
<html>
<head>
<title>www.unixmen1.local</title>
</head>
<body>
<h1>Welcome To Unixmen1.local website</h1>
</body>
</html>
```

Save and close the file.

Similarly, add the sample page to the second virtual host.

```
sudo vi /var/www/html/unixmen2.local/public_html/index.html
```

Add the following contents:

```
<html>
<head>
<title>www.unixmen2.local</title>
</head>
<body>
<h1>Welcome To Unixmen2.local website</h1>
</body>
</html>
```

Save and close the file.

5. Create Virtual Host Files

By default, Apache comes with a default virtual host file called **000-default.conf**. We will copy the 000-default.conf file contents to our new virtual host files.

```
sudo cp /etc/apache2/sites-available/000-default.conf /etc/apache2/sites-
available/unixmen1.local.conf
```

```
sudo cp /etc/apache2/sites-available/000-default.conf /etc/apache2/sites-
available/unixmen2.local.conf
```

Make sure the virtual host files contains **.conf** extension at the end.

Now, modify the unixmen1.local.conf file to reflect with our new own values.

```
sudo vi /etc/apache2/sites-available/unixmen1.local.conf
```

Make the relevant changes that reflect to the unixmen1 site.

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port
    that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@unixmen1.local
    ServerName unixmen1.local
    ServerAlias www.unixmen1.local
    DocumentRoot /var/www/html/unixmen1.local/public_html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>
```

Like wise, modify the second virtual host file.

```
sudo vi /etc/apache2/sites-available/unixmen2.local.conf
```

Make the relevant changes that reflect to the unixmen2 site.

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port
    that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@unixmen2.local
    ServerName unixmen2.local
    ServerAlias www.unixmen2.local
    DocumentRoot /var/www/html/unixmen2.local/public_html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
```

```

LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf
</VirtualHost>

```

After modifying the virtual hosts files, disable the default virtual host (000.default.conf), and enable new virtual hosts as shown below.

```
sudo a2dissite 000-default.conf
```

```
sudo a2ensite unixmen1.local.conf
```

```
sudo a2ensite unixmen2.local.conf
```

Finally, restart the apache service.

In Ubuntu 15.10/15.04:

```
sudo systemctl restart apache2
```

In Ubuntu 14.10 and earlier versions:

```
sudo service apache2 restart
```

That's it. Now, we successfully configured the apache virtual hosts on our Ubuntu server.

Testing Virtual Hosts

Edit file **/etc/hosts**,

```
sudo vi /etc/hosts
```

Add the virtual domain names one by one as shown below.

```

[...]
192.168.1.103    unixmen1.local
192.168.1.103    unixmen2.local

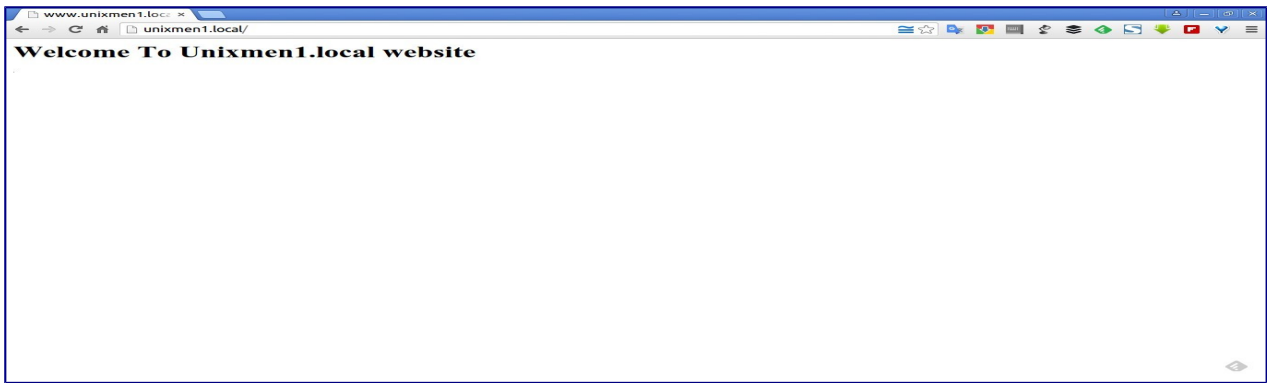
```

Save and close the file.

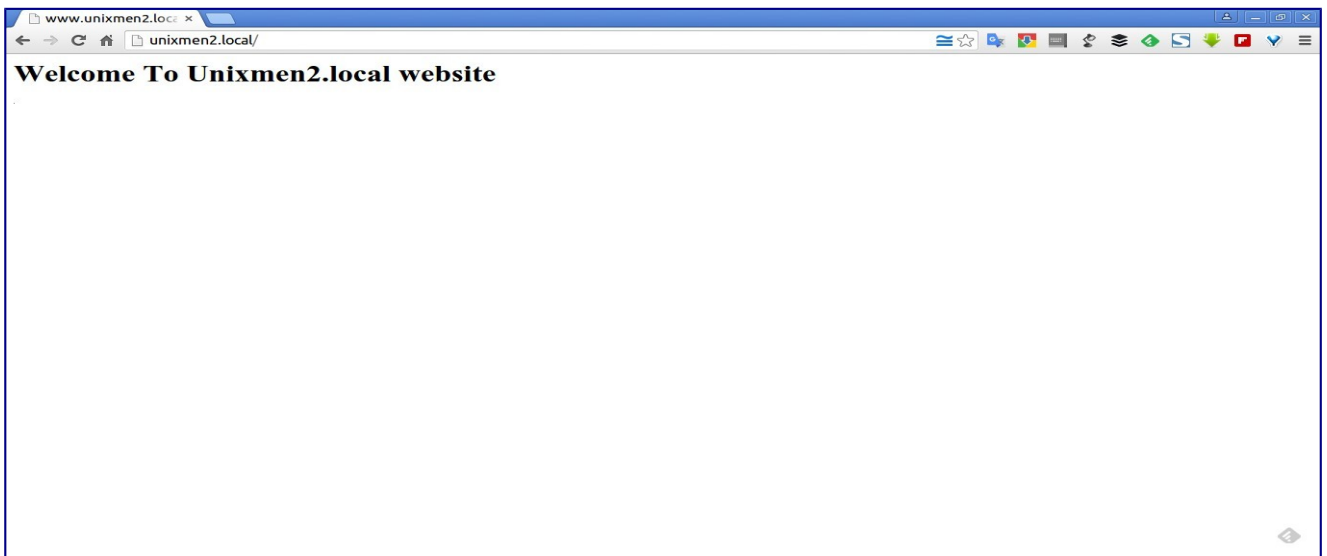
Note: In case, you want to access the above sites from a remote system, you need to add the above two lines to your remote system's **/etc/hosts/** file as well. Don't forget to replace the IP address with your own.

Open up your browser and point to the URL **http://unixmen1.local** or **http://unixmen2.local**. You should see the sample pages which we created earlier.

Unixmen1.local Test page:



Unixmen2.local Test page:



Alias and ScriptAlias Statements

Directory definitions are sometimes preceded by "Alias" or "ScriptAlias" statements. Alias maps a url path to a directory path.

ScriptAlias operates in the same way, but is used to define directories that will have executable components in them.

For instance, this line in a Virtual Host that handles request to "example.com" would allow access to content within "/path/to/content/" by navigating to "example.com/content/":

```
Alias /content/ /path/to/content/
```

Following the alias, you should remember to define the directory with access privileges as discussed in the previous section.

Enabling Sites and Modules in Apache

Once you have a Virtual Host file that meets your requirements, you can use the tools included with Apache to transition them into live sites.

To automatically create a symbolic link in the "sites-enabled" directory to an existing file in the "sites-available" directory, issue the following command:

```
sudo a2ensite virtual_host_file_name
```

After enabling a site, issue the following command to tell Apache to re-read its configuration files, allowing the change to propagate:

```
sudo service apache2 reload
```

There is also a companion command for disabling a Virtual Host. It operates by removing the symbolic link from the "sites-enabled" directory:

```
sudo a2dissite virtual_host_file_name
```

Again, reload the configuration to make the change happen:

```
sudo service apache2 reload
```

Modules can be enabled or disabled by using the "a2enmod" and "a2dismod" commands respectively. They work in the same way as the "site" versions of these commands.

Remember to reload your configuration changes after modules have been enabled or disabled as well.

How To Create a SSL Certificate on Apache for Ubuntu 14.04

Introduction

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.

In this guide, we'll cover how to create a **self-signed SSL certificate** for Apache on an Ubuntu 14.04 server, which will allow you to encrypt traffic to your server. While this does not provide the benefit of third party validation of your server's identity, it fulfills the requirements of those simply wanting to transfer information securely.

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 management. 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 nano /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.



The site's security certificate is not trusted!

You attempted to reach **107.170.75.175**, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

You should not proceed, **especially** if you have never seen this warning before for this site.

Proceed anyway

Back to safety

▶ [Help me understand](#)

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:

The screenshot shows a web browser window with a security warning dialog box open. The address bar shows a URL starting with `https://107.170.75.175`. The dialog box has two tabs: "Permissions" and "Connection". The "Connection" tab is active, showing a warning icon and the text: "The identity of this website has not been verified." Below this, it says "Server's certificate is not trusted." and provides a link to "Certificate Information".

The dialog box also shows a green lock icon and the text: "Your connection to 107.170.75.175 is encrypted with 128-bit encryption." It further states: "The connection uses TLS 1.2." and "The connection is encrypted and authenticated using AES_128_GCM and uses ECDHE_RSA as the key exchange mechanism."

Below the connection information, there is a section titled "Site information" with an information icon. It says: "You have never visited this site before today." and provides a link to "What do these mean?".

In the background, a terminal window is visible, showing the following configuration commands:

```
-- *.conf
-- conf-enabled
-- *.conf
-- sites-enabled
-- *.conf
```

On the right side of the browser window, a webpage titled "e2 Ubuntu D" is partially visible. It features a red banner that says "It works!" and text that reads: "d to test the correct oper: based on the equivalent I an read this page, it mea should **replace this file** (I TP server. ite and don't know what due to maintenance. If th". Below this, there is a section titled "Configuration Overvi" and text that says: "on is different from the u tion with Ubuntu tools. T **pache2/README.Debi** he web server itself can b n this server. e2 web server installio".