Lab 1: Apache Web Server Installation & Maintenance

Objectives:

- To install, administer and maintain an Apache web server.

Submission:

- Checkpoints that need to be shown to the course teacher.

Instruction:

In this lab, you will install, administer and maintain an Apache web server in your Ubuntu machine. Apache is the most widely-used web servers in the world. In fact, it is one of the most widely used open source software in the world!

Many of you might have already used Apache web server under the LAMP stack. However, most of you used the Apache server under the LAMP stack as it was, without understanding what was happening underneath. In this lab, we will do things in a different way so that you know how to control a web server and how to hosts different websites in a single web server.

Task-1: Setting up Apache

The following is a tutorial collected from: https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-18-04.

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.

During installation, Apache registers itself with UFW to provide a few application profiles that can be used to enable or disable access to Apache through the firewall.

List the *ufw* application profiles by typing:

- *sudo ufw app list*

You will see a list of the application profiles:

```
Output
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

As you can see, there are three profiles available for Apache:

- **Apache**: This profile opens only port 80 (normal, unencrypted web traffic)
- **Apache Full**: This profile opens both port 80 (normal, unencrypted web traffic) and port 443 (TLS/SSL encrypted traffic)
- **Apache Secure**: This profile opens only port 443 (TLS/SSL encrypted traffic)

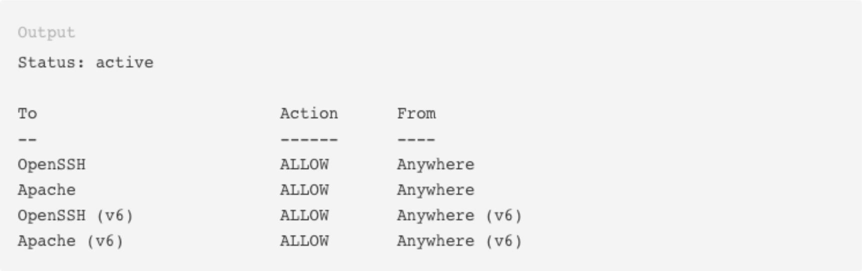
It is recommended that you enable the most restrictive profile that will still allow the traffic you've configured. Since we haven't configured SSL for our server yet in this guide, we will only need to allow traffic on port 80:

- *sudo ufw allow 'Apache'*

You can verify the change by typing:

- sudo ufw status

You should see HTTP traffic allowed in the displayed output:



```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
Apache (v6)                ALLOW       Anywhere (v6)
```

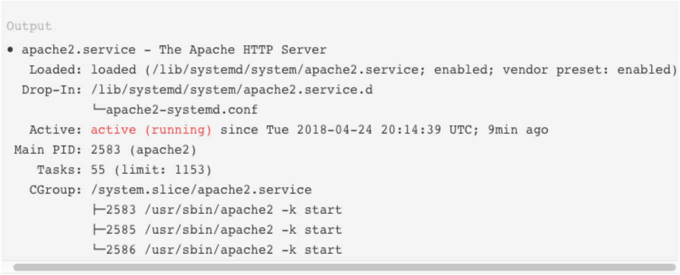
As you can see, the profile has been activated to allow access to the web server.

Step 3 — Checking your Web Server

At the end of the installation process, Ubuntu 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



```
Output
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Tue 2018-04-24 20:14:39 UTC; 9min ago
 Main PID: 2583 (apache2)
    Tasks: 55 (limit: 1153)
   CGroup: /system.slice/apache2.service
           ├─2583 /usr/sbin/apache2 -k start
           ├─2585 /usr/sbin/apache2 -k start
           └─2586 /usr/sbin/apache2 -k start
```

As you can see from this output, the service appears to have started successfully. However, the best way to test this is to request a page from Apache.

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):

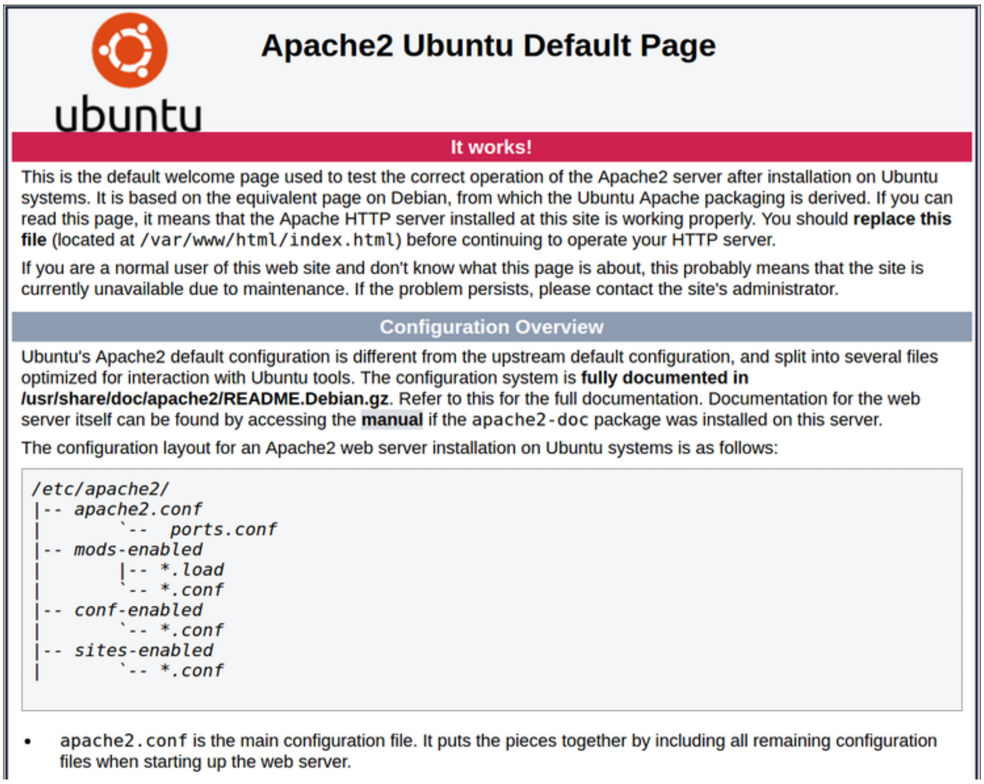
- *127.0.0.1 webserverlab.com*

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

In Ubuntu, you can retrieve the IP address of your machine using the **ifconfig** command in the console.

You should see the default Apache web page:



Apache2 Ubuntu Default Page

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.

This page indicates that Apache is working correctly. It also includes some basic information about important Apache files and directory locations.

Checkpoint – 1: Show this to your course teacher. (3 Marks)

Task-2: Setting up virtual hosts

Step 1 — Managing the Apache Process

Now that you have your web server up and running, let's go over some basic management commands.

To stop your web server, type:

- *sudo systemctl stop apache2*

To start the web server when it is stopped, type:

- *sudo systemctl start apache2*

To stop and then start the service again, type:

- *sudo systemctl restart apache2*

If you are simply making configuration changes, Apache can often reload without dropping connections. To do this, use this command:

- *sudo systemctl reload apache2*

By default, Apache is configured to start automatically when the server boots. If this is not what you want, disable this behaviour by typing:

- *sudo systemctl disable apache2*

To re-enable the service to start up at boot, type:

- *sudo systemctl enable apache2*

Apache should now start automatically when the server boots again.

Step 2 — Setting up a single virtual host

When using the Apache web server, you can use *virtual hosts* (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. We will set up another domain called **example.com**. At first, add the **example.com** domain in your /etc/hosts file as you did previously.

Apache generally has one server block enabled by default that is configured to serve documents from the /var/www/html directory. While this works well for a single site, it can become unwieldy if you are hosting multiple sites. Instead of modifying /var/www/html, let's create a directory structure within /var/www for our **example.com** site, leaving /var/www/html in place as the default directory to be served if a client request doesn't match any other sites.

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 environmental 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 favourite editor:

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

Inside, add the following sample HTML:

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

```
<html>
   <head>
      <title>Welcome to Example.com!</title>
   </head>
   <body>
      <h1>Success!  The example.com server block is working!</h1>
   </body>
</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:

/etc/apache2/sites-available/example.com.conf

```
<VirtualHost *:80>
   ServerAdmin admin@example.com
   ServerName example.com
   ServerAlias www.example.com
   DocumentRoot /var/www/example.com/html
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

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:

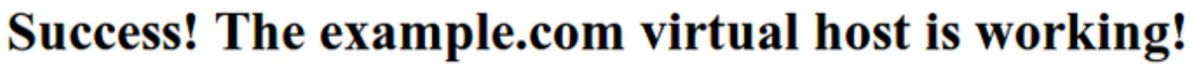
- *sudo apache2ctl configtest*

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. You can test this by navigating to http://example.com, where you should see something like this:



Success! The example.com virtual host is working!

Checkpoint – 2: Show this to your course teacher. (6 Marks)

Step 3 — Setting Up multiple virtual hosts

Repeat the steps described above to setup another virtual host for the domain **anothervhost.com** with a different html file containing different contents.

Checkpoint – 3: Show this to your course teacher. (6 Marks)

Other steps – Getting Familiar with Important Apache Files and Directories (Not evaluated)

Now that you know how to manage the Apache service itself, you should take a few minutes to familiarise yourself with a few important directories and files. Do not spend more than ten minutes on this. Keep it for future references.

Content

- /var/www/html: The actual web content, which by default only consists of the default Apache page you saw earlier, is served out of the /var/www/html directory. This can be changed by altering Apache configuration files.

Server Configuration

- /etc/apache2: The Apache configuration directory. All of the Apache configuration files reside here.
- /etc/apache2/apache2.conf: The main Apache configuration file. This can be modified to make changes to the Apache global configuration. This file is responsible for loading many of the other files in the configuration directory.
- /etc/apache2/ports.conf: This file specifies the ports that Apache will listen on. By default, Apache listens on port 80 and additionally listens on port 443 when a module providing SSL capabilities is enabled.
- /etc/apache2/sites-available/: The directory where per-site virtual hosts can be stored. Apache will not use the configuration files found in this directory unless they are linked to the sites-enabled directory. Typically, all server block configuration is done in this directory, and then enabled by linking to the other directory with the a2ensite command.
- /etc/apache2/sites-enabled/: The directory where enabled per-site virtual hosts are stored. Typically, these are created by linking to configuration files found in the sites-availabledirectory with the a2ensite. Apache reads the configuration files and links found in this directory when it starts or reloads to compile a complete configuration.
- /etc/apache2/conf-available/, /etc/apache2/conf-enabled/: These directories have the same relationship as the sites-available and sites-enabled directories, but are

used to store configuration fragments that do not belong in a virtual host. Files in the conf-available directory can be enabled with the a2enconf command and disabled with the a2disconf command.

- /etc/apache2/mods-available/, /etc/apache2/mods-enabled/: These directories contain the available and enabled modules, respectively. Files in ending in .load contain fragments to load specific modules, while files ending in .conf contain the configuration for those modules. Modules can be enabled and disabled using the a2enmod and a2dismod command.

Server Logs

- /var/log/apache2/access.log: By default, every request to your web server is recorded in this log file unless Apache is configured to do otherwise.
- /var/log/apache2/error.log: By default, all errors are recorded in this file. The LogLeveldirective in the Apache configuration specifies how much detail the error logs will contain.