# **WEB STACK IMPLEMENTATION (LAMP STACK)**

## **Welcome to your very first PBL project implementation**

As you kick off your career in DevOps, you will soon realise that everything you will be doing as a DevOps engineer is around software, websites, applications etc. And, there are different stack of technologies that make different solutions possible. These stack of technologies are regarded as **WEB STACKS**. Examples of Web Stacks include **LAMP**, **LEMP**, **MEAN**, and **MERN** stacks. As you begin your journey, you will be required to understand, and implement 1 of these stacks all by yourself.

### **After successful completion of PBL projects 1 to 8 (Starter and Progressive Projects), you will be able to achieve the following.**

1. Become very confident on the Linux Terminal
2. Deep Understanding of Web Stacks and familiarity between the differences between the different Web Technology stacks such as **LAMP**, **LEMP**, **MEAN**, and **MERN** stacks
3. Solid Linux administration skills in Storage management, NFS, troubleshooting, and basic networking.

Being able to work on Linux requires the ability to work outside the level of your present knowledge. What this means is that, even in the real world, you will be faced with tasks that you have never worked on before, and with **Google** search and its results, you can achieve a lot. Thanks to “**Google!!!**”. The ability to construct your querry to Google correctly goes a long way in determining how fast you get relevant information required to complete your tasks.

## **Research Work**

1. Conduct a google search on what software development life cycle (SDLC) is and document your finding in a Google word file.
2. Conduct another google search, understand what LAMP stack means.

## **Instructions On How To Submit Your Work For Review And Feedback**

Follow the steps below to submit your work for review

1. Create a word document on Google Drive
2. Write down the steps you took to accomplish your work
3. Make screenshots wherever possible.
   1. On Windows you can use Snipping Tool
   2. On Mac you can use Lightshot
4. Name your document according to the Project work
5. Click the Submit your project for review in DAREY.IO project dashboard, and paste the link to your project, and submit.

## **Project Prerequisites**

In order to complete this project, you will need to have an Ubuntu 20.04 server installed in your Virtual Box software.

1. [Here is how to configure Virtual Box](https://www.youtube.com/watch?v=wql6adU2JeE&list=PLtPuNR8I4TvkwU7Zu0l0G_uwtSUXLckvh&index=3)
2. [Here is how to install and configure Ubuntu 20.04](https://www.youtube.com/watch?v=Lw-VAYWJumo&list=PLtPuNR8I4TvkwU7Zu0l0G_uwtSUXLckvh&index=4)

## **Step 1 — Installing Apache and Updating the Firewall**

What exactly is Apache? Apache is the most widely used web server software. Developed and maintained by Apache Software Foundation, Apache is an open source software available for free. It runs on 67% of all webservers in the world. It is fast, reliable, and secure. It can be highly customized to meet the needs of many different environments by using extensions and modules. Most WordPress hosting providers use Apache as their web server software. However, websites and other applications can run on other web server software as well. Such as Nginx, Microsoft’s IIS, etc.

The Apache web server is among the most popular web servers in the world. It’s well documented, has an active community of users, and has been in wide use for much of the history of the web, which makes it a great default choice for hosting a website.

Install Apache using Ubuntu’s package manager, apt:

$ sudo apt update

$ sudo apt install apache2

If this is the first time you’re using sudo within this session, you’ll be prompted to provide your user’s password to confirm you have the right privileges to manage system packages with apt. You’ll also be prompted to confirm Apache’s installation by pressing Y, then ENTER.

Once the installation is finished, you’ll need to adjust your firewall settings to allow HTTP traffic. UFW has different application profiles that you can leverage for accomplishing that. To list all currently available UFW application profiles, you can run:

$ sudo ufw app list

You’ll see output like this:

Output

Available applications:

Apache

Apache Full

Apache Secure

OpenSSH

Here’s what each of these profiles mean:

* **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).

For now, it’s best to allow only connections on port 80, since this is a fresh Apache installation and you still don’t have a TLS/SSL certificate configured to allow for HTTPS traffic on your server.

To only allow traffic on port 80, use the Apache profile:

$ sudo ufw allow in "Apache"

You can verify the change with:

$ sudo ufw status

Output

Status: active

To Action From

-- ------ ----

OpenSSH ALLOW Anywhere

Apache ALLOW Anywhere

OpenSSH (v6) ALLOW Anywhere (v6)

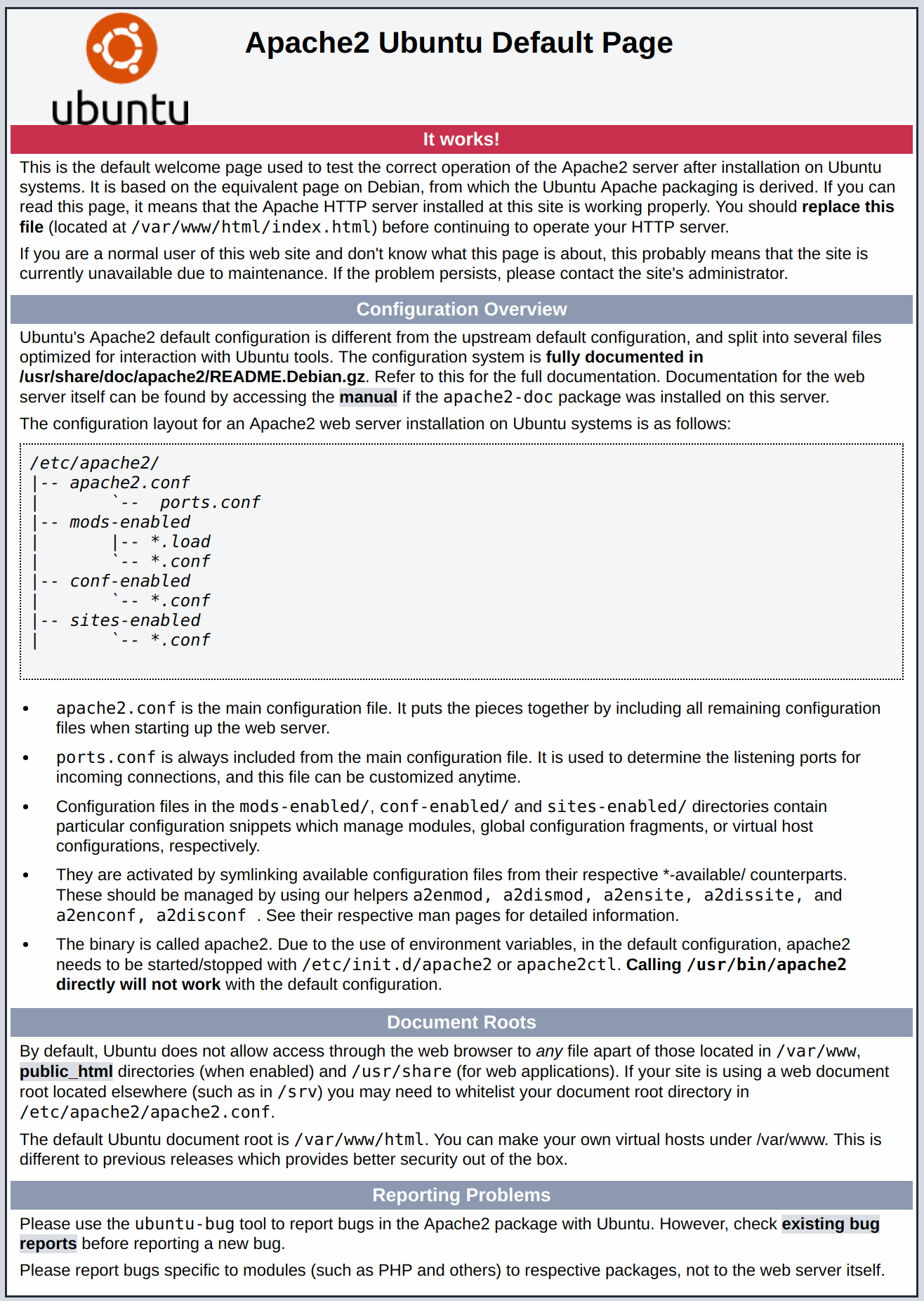
Apache (v6) ALLOW Anywhere (v6)

Traffic on port 80 is now allowed through the firewall.

You can do a spot check right away to verify that everything went as planned by visiting your server’s public IP address in your web browser (see the note under the next heading to find out what your public IP address is if you do not have this information already):

http://your\_server\_ip

You’ll see the default Ubuntu 20.04 Apache web page, which is there for informational and testing purposes. It should look something like this:



If you see this page, then your web server is now correctly installed and accessible through your firewall.

## **How To Find your Server’s Public IP Address**

If you do not know what your server’s public IP address is, there are a number of ways you can find it. Usually, this is the address you use to connect to your server through SSH.

There are a few different ways to do this from the command line. First, you could use the iproute2 tools to get your IP address by typing this:

$ ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.\*$//'

This will give you two or three lines back. They are all correct addresses, but your computer may only be able to use one of them, so feel free to try each one.

## **Step 2 — Installing MySQL**

Now that you have a web server up and running, you need to install the database system to be able to store and manage data for your site. MySQL is a popular database management system used within PHP environments.

Again, use apt to acquire and install this software:

$ sudo apt install mysql-server

When prompted, confirm installation by typing Y, and then ENTER.

When the installation is finished, it’s recommended that you run a security script that comes pre-installed with MySQL. This script will remove some insecure default settings and lock down access to your database system. Start the interactive script by running:

$ sudo mysql\_secure\_installation

This will ask if you want to configure the VALIDATE PASSWORD PLUGIN.

**Note**: Enabling this feature is something of a judgment call. If enabled, passwords which don’t match the specified criteria will be rejected by MySQL with an error. It is safe to leave validation disabled, but you should always use strong, unique passwords for database credentials.

Answer Y for yes, or anything else to continue without enabling.

VALIDATE PASSWORD PLUGIN can be used to test passwords

and improve security. It checks the strength of password

and allows the users to set only those passwords which are

secure enough. Would you like to setup VALIDATE PASSWORD plugin?

Press y|Y for Yes, any other key for No:

If you answer “yes”, you’ll be asked to select a level of password validation. Keep in mind that if you enter 2 for the strongest level, you will receive errors when attempting to set any password which does not contain numbers, upper and lowercase letters, and special characters, or which is based on common dictionary words.

There are three levels of password validation policy:

LOW Length >= 8

MEDIUM Length >= 8, numeric, mixed case, **and** special characters

STRONG Length >= 8, numeric, mixed case, special characters **and** dictionary file

Please enter 0 = LOW, 1 = MEDIUM **and** 2 = STRONG: 1

Regardless of whether you chose to set up the VALIDATE PASSWORD PLUGIN, your server will next ask you to select and confirm a password for the MySQL **root** user. This is not to be confused with the **system root**. The **database root** user is an administrative user with full privileges over the database system. Even though the default authentication method for the MySQL root user dispenses the use of a password, **even when one is set**, you should define a strong password here as an additional safety measure. We’ll talk about this in a moment.

If you enabled password validation, you’ll be shown the password strength for the root password you just entered and your server will ask if you want to continue with that password. If you are happy with your current password, enter Y for “yes” at the prompt:

Estimated strength of the password: 100

Do you wish to continue with the password provided?(Press y|Y for Yes, any other key for No) : y

For the rest of the questions, press Y and hit the ENTER key at each prompt. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes you have made.

When you’re finished, test if you’re able to log in to the MySQL console by typing:

$ sudo mysql

This will connect to the MySQL server as the administrative database user **root**, which is inferred by the use of sudo when running this command. You should see output like this:

Welcome to the MySQL monitor. Commands end **with** ; **or** \g.

Your MySQL connection id **is** 22

Server version: 8.0.19-0ubuntu5 (Ubuntu)

Copyright (c) 2000, 2020, Oracle **and**/**or** its affiliates. All rights reserved.

Oracle **is** a registered trademark of Oracle Corporation **and**/**or** its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' **or** '\h' **for** help. Type '\c' to clear the current input statement.

mysql>

To exit the MySQL console, type:

mysql> exit

Notice that you didn’t need to provide a password to connect as the **root** user, even though you have defined one when running the mysql\_secure\_installation script. That is because the default authentication method for the administrative MySQL user is unix\_socket instead of password. Even though this might look like a security concern at first, it makes the database server more secure because the only users allowed to log in as the **root** MySQL user are the system users with sudo privileges connecting from the console or through an application running with the same privileges. In practical terms, that means you won’t be able to use the administrative database **root** user to connect from your PHP application. Setting a password for the **root** MySQL account works as a safeguard, in case the default authentication method is changed from unix\_socket to password.

For increased security, it’s best to have dedicated user accounts with less expansive privileges set up for every database, especially if you plan on having multiple databases hosted on your server.

**Note**: At the time of this writing, the native MySQL PHP library mysqlnd doesn’t support caching\_sha2\_authentication, the default authentication method for MySQL 8. For that reason, when creating database users for PHP applications on MySQL 8, you’ll need to make sure they’re configured to use mysql\_native\_password instead. We’ll demonstrate how to do that in Step 6.

Your MySQL server is now installed and secured. Next, we’ll install PHP, the final component in the LAMP stack.

## **Step 3 — Installing PHP**

You have Apache installed to serve your content and MySQL installed to store and manage your data. PHP is the component of our setup that will process code to display dynamic content to the final user. In addition to the php package, you’ll need php-mysql, a PHP module that allows PHP to communicate with MySQL-based databases. You’ll also need libapache2-mod-php to enable Apache to handle PHP files. Core PHP packages will automatically be installed as dependencies.

To install these packages, run:

$ sudo apt install php libapache2-mod-php php-mysql

Once the installation is finished, you can run the following command to confirm your PHP version:

php -v

PHP 7.4.3 (cli) (built: Mar 26 2020 20:24:23) ( NTS )

Copyright (c) The PHP Group

Zend Engine v3.4.0, Copyright (c) Zend Technologies

**with** Zend OPcache v7.4.3, Copyright (c), by Zend Technologies

At this point, your LAMP stack is fully operational, but before you can test your setup with a PHP script, it’s best to set up a proper Apache Virtual Host to hold your website’s files and folders. We’ll do that in the next step.

## **Step 4 — Creating a Virtual Host for your Website using Apache**

When using the Apache web server, you can create virtual hosts to encapsulate configuration details and host more than one domain from a single server. In this project, you will set up a domain called propitixhomes.local, but you can replace this with any domain of your choice.

Apache on Ubuntu 20.04 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**, we’ll create a directory structure within **/var/www** for the propitixhomes.local 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 propitixhomes.local as follows:

$ sudo mkdir /var/www/propitixhomes.local

Next, assign ownership of the directory with the $USER environment variable, which will reference your current system user:

$ sudo chown -R $USER:$USER /var/www/propitixhomes.local

Then, open a new configuration file in Apache’s sites-available directory using your preferred command-line editor. Here, we’ll be using vi or vim (They are the same by the way):

$ sudo vi /etc/apache2/sites-available/propitixhomes.local.conf

This will create a new blank file. Paste in the following bare-bones configuration by hitting on i on the keyboard to enter the insert mode, and paste the text:

<VirtualHost \*:80>

ServerName propitixhomes.local

ServerAlias www.propitixhomes.local

ServerAdmin webmaster@localhost

DocumentRoot /var/www/propitixhomes.local

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

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

</VirtualHost>

To save and close the file, simply follow the steps below:

1. Hit the esc button on the keyboard
2. Type :
3. Type wq. **w** for write and **q** for quit
4. Hit ENTER to save the file

You can use the ls command to show the new file in the **sites-available** directory

$ sudo ls /etc/apache2/sites-available

With this VirtualHost configuration, we’re telling Apache to serve propitixhomes.local using **/var/www/propitixhomes.local** as the web root directory. If you’d like to test Apache without a domain name, you can remove or comment out the options ServerName and ServerAlias by adding a **#** character in the beginning of each option’s lines. Adding the **#** character there will tell the program to skip processing the instructions on those lines.

You can now use a2ensite to enable the new virtual host:

$ sudo a2ensite propitixhomes.local

You might want to disable the default website that comes installed with Apache. This is required if you’re not using a custom domain name, because in this case Apache’s default configuration would overwrite your virtual host. To disable Apache’s default website, type:

$ sudo a2dissite 000-default

To make sure your configuration file doesn’t contain syntax errors, run:

$ sudo apache2ctl configtest

Finally, reload Apache so these changes take effect:

$ sudo systemctl reload apache2

Your new website is now active, but the web root **/var/www/propitixhomes.local** is still empty. Create an index.html file in that location so that we can test that the virtual host works as expected:

vi /var/www/propitixhomes.local/index.html

Now type anything in the file and save it. (Remember how you saved the file using vim?)

Now go to your browser and access your server’s domain name or IP address once again:

http://server\_domain\_or\_IP

If you see the text you wrote in the file, then it means your Apache virtual host is working as expected.

You can leave this file in place as a temporary landing page for your application until you set up an index.php file to replace it. Once you do that, remember to remove or rename the index.html file from your document root, as it would take precedence over an index.php file by default.

## **A Note About DirectoryIndex on Apache**

With the default **DirectoryIndex** settings on Apache, a file named index.html will always take precedence over an index.php file. This is useful for setting up maintenance pages in PHP applications, by creating a temporary index.html file containing an informative message to visitors. Because this page will take precedence over the index.php page, it will then become the landing page for the application. Once maintenance is over, the index.html is renamed or removed from the document root, bringing back the regular application page.

In case you want to change this behavior, you’ll need to edit the **/etc/apache2/mods-enabled/dir.conf** file and modify the order in which the **index.php** file is listed within the **DirectoryIndex** directive:

sudo vim /etc/apache2/mods-enabled/dir.conf

<IfModule mod\_dir.c>

DirectoryIndex index.php index.html index.cgi index.pl index.xhtml index.htm

</IfModule>

After saving and closing the file, you’ll need to reload Apache so the changes take effect:

$ sudo systemctl reload apache2

In the next step, we’ll create a PHP script to test that PHP is correctly installed and configured on your server.

Now that you have a custom location to host your website’s files and folders, we’ll create a PHP test script to confirm that Apache is able to handle and process requests for PHP files.

Create a new file named info.php inside your custom web root folder:

$ vim /var/www/propitixhomes.local/info.php

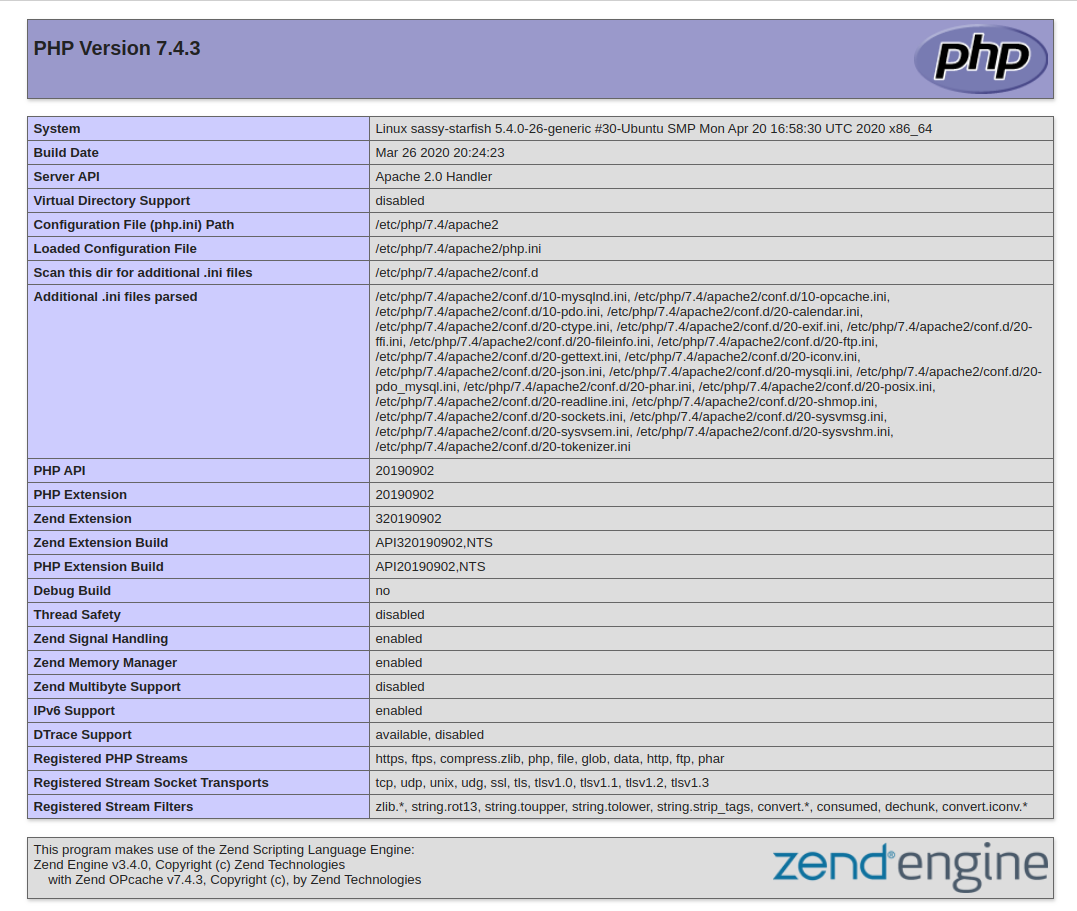
This will open a blank file. Add the following text, which is valid PHP code, inside the file:

<?php

phpinfo();

When you are finished, save and close the file.

You’ll see a page similar to this:



This page provides information about your server from the perspective of PHP. It is useful for debugging and to ensure that your settings are being applied correctly.

If you can see this page in your browser, then your PHP installation is working as expected.

After checking the relevant information about your PHP server through that page, it’s best to remove the file you created as it contains sensitive information about your PHP environment -and your Ubuntu server. You can use rm to do so:

$ sudo rm /var/www/propitixhomes.local/info.php

You can always recreate this page if you need to access the information again later.

Credit: [This guide was inspired by Digital Ocean](https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-20-04#step-3-%E2%80%94-installing-php)