How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 20.04.3

Introduction

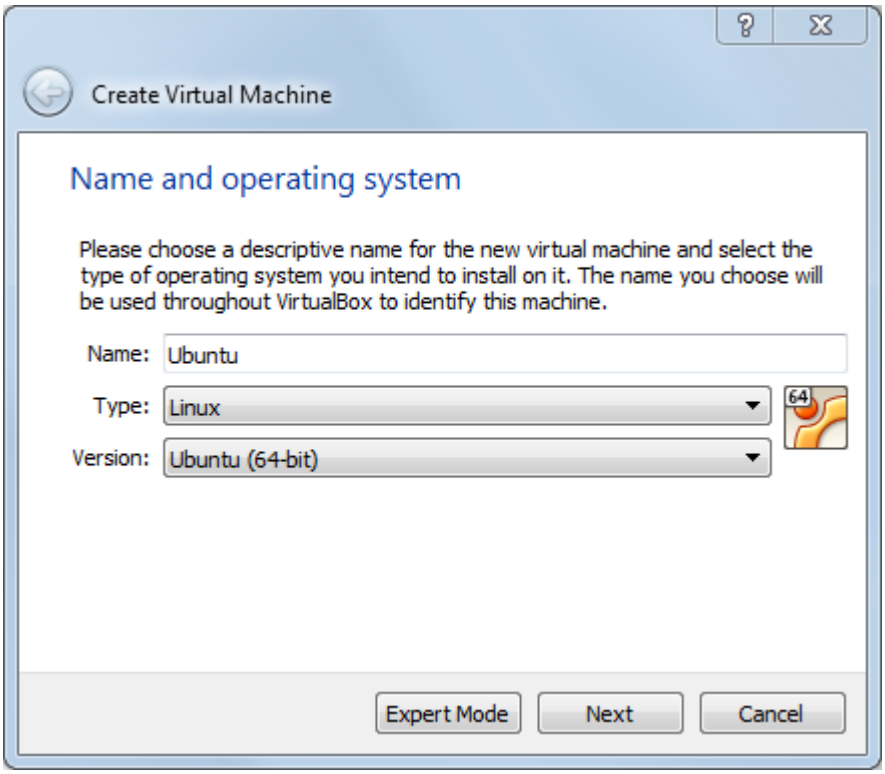
A "LAMP" stack is a group of open source software that is typically installed together to enable a server to host dynamic websites and web apps. This term is actually an acronym which represents the **L**inux operating system, with the **A**pache web server. The site data is stored in a **M**ySQL database, and dynamic content is processed by **P**HP.

Step 1: Install Ubuntu Linux

Ubuntu is a free open source distribution of Linux. First, download Ubuntu version 20.04.3 LTS from here https://ubuntu.com/download/desktop

Instead of installing Ubuntu directly on the native PC, we will install Ubuntu in a virtual machine using Oracles' Virtual Box. You can download Virtual Box from here https://www.virtualbox.org/wiki/Downloads

Open Virtual Box, and click on the blue New icon to create a new virtual machine for Ubuntu 64-bit.



Use all default settings.

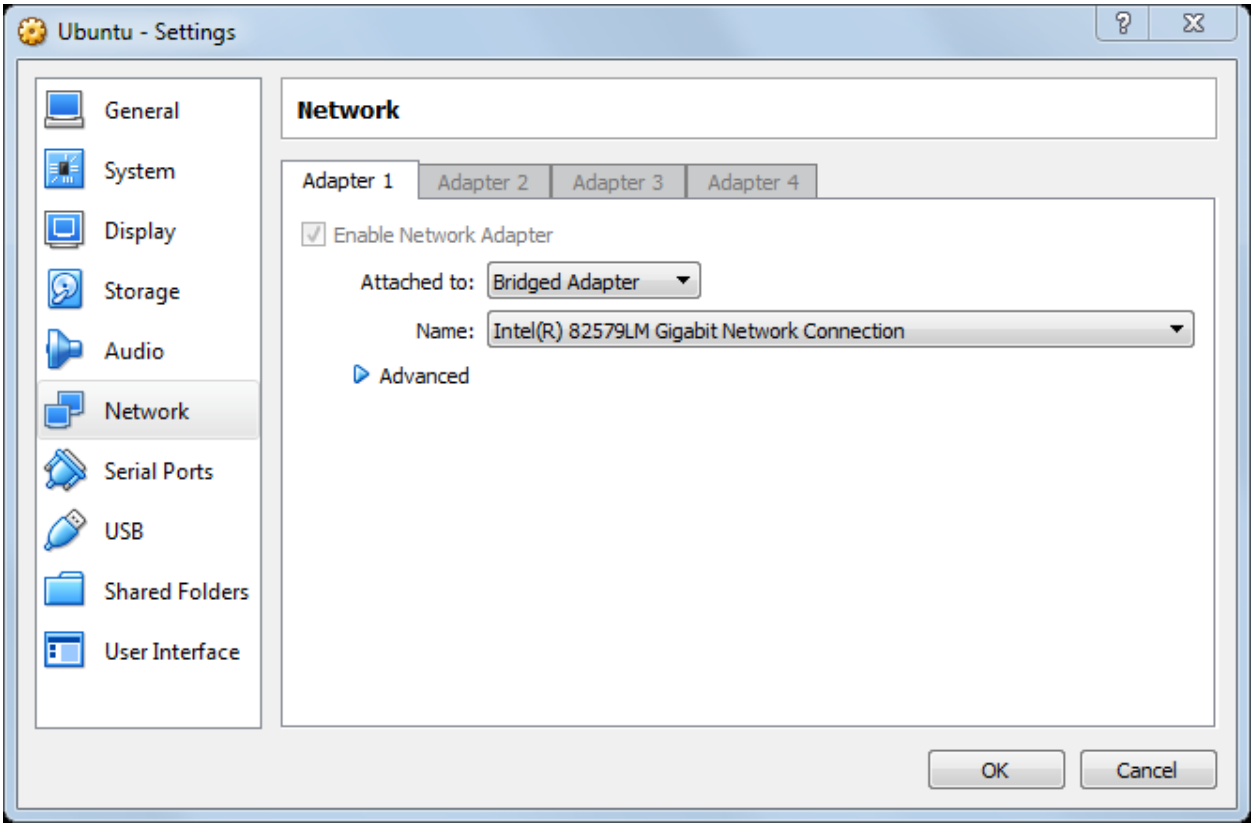
Click on the green Start icon to run (“power up”) the machine. Select the location for your Ubuntu installation media.

When it asks for the root password, use **lsucs** for the password.

After Ubuntu has successfully installed, from the Oracle Virtual Box window, click on Devices in the menu bar and select **Insert Guest Addition CD Image** to add this CD image.

Under the Devices menu, you can also enable the Shared Clipboard and Drag & Drop between Windows and Ubuntu.

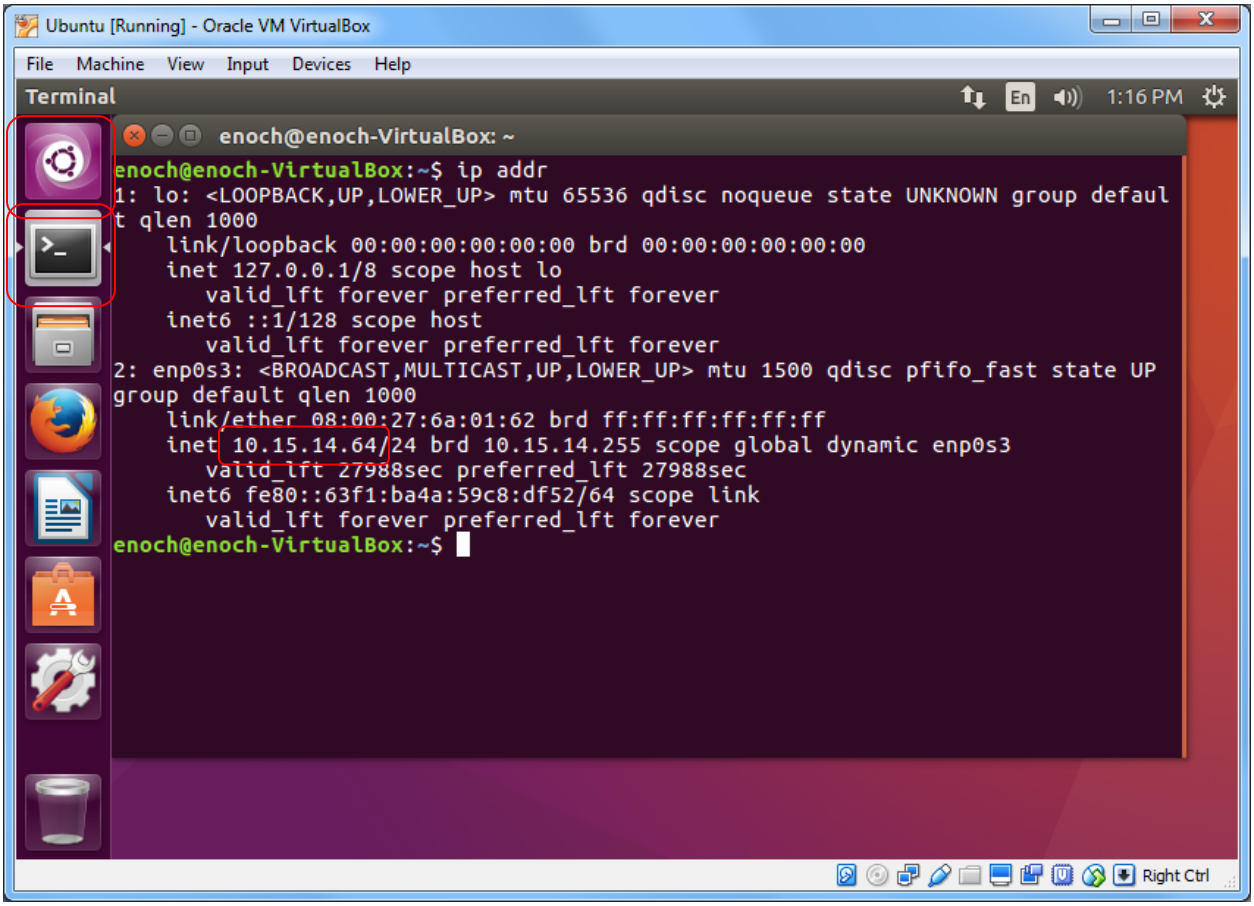
After Ubuntu has successfully installed, from the Oracle Virtual Box window, click on Devices in the menu bar and select Network, then Network Settings. In the window that opens up, select **Bridged Adapter** to Attached to. Select your network card.



In Ubuntu, open up a Linux Terminal window. If the Terminal window icon is not in the sidebar then you can do a search for Terminal. See picture below for the Terminal icon and the Application Search icon.

Type in the following command to find out your computer's IP address.

```
$ ip addr
```

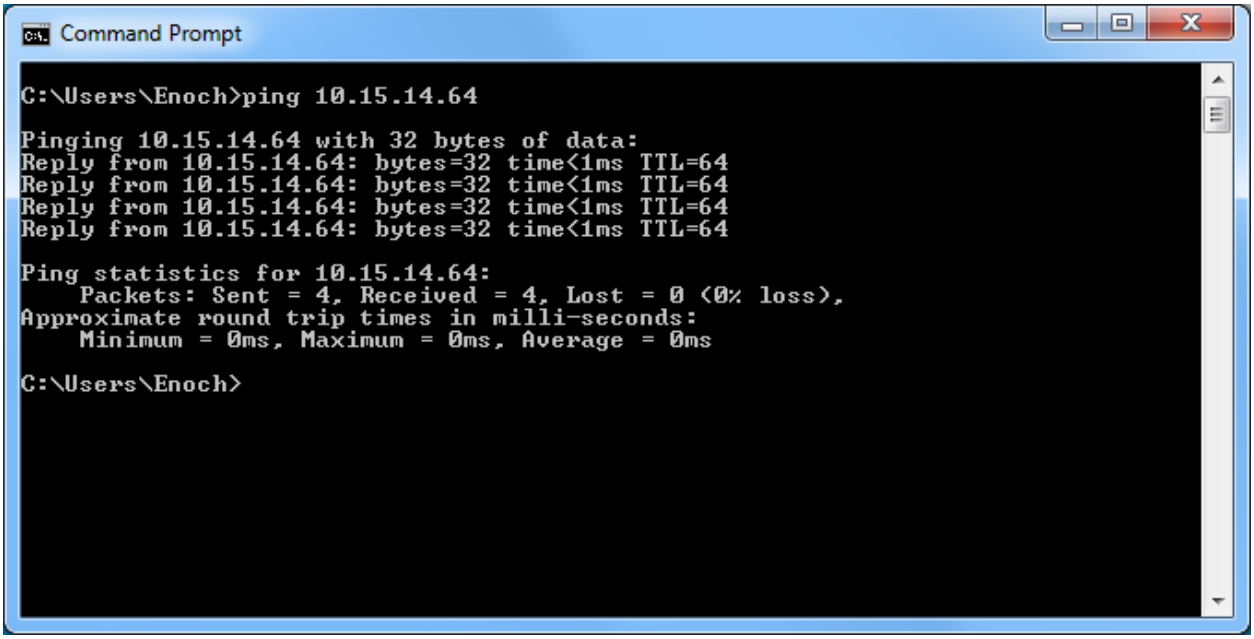


Your IP address should be 10.15.15.x if you're in PSC147. If you're at home it will probably be something like 192.168.1.x. If not, your network adaptor was not setup correctly. Turn off your Linux machine and exit your Virtual Box, and then restart everything.

Now from another computer, open up a Windows command prompt window, or a Linux Terminal window and type in the following **ping** command but replace the IP address with the one that you obtained above.

```
C:\Users\Majors> ping 10.15.14.64
```

You should receive a reply similar to the following. Press **Ctrl-C** to stop the replies.



You can also try the following command to see if you can connect to Google. Might need to add the -4 because your install defaults to IPv6 but need ping Google with IPv4.

```
C:\Users\Majors> ping -4 google.com
```

Open up the FireFox web browser and try to browse to some of your favorite websites to make sure that your internet connection is working.

Step 2: Install Apache and Allow in Firewall

The Apache web server is among the most popular web servers in the world. It's well-documented, 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.

We can install Apache easily using Ubuntu's package manager, `apt`. A package manager allows us to install most software pain-free from a repository maintained by Ubuntu.

For our purposes, we can get started by typing the following commands.

```
$ sudo apt-get update
$ sudo dpkg --configure -a
$ sudo apt-get install apache2
```

Since we are using a `sudo` command, these operations get executed with root privileges. It will ask you for your regular user's password to verify your intentions.

Once you've entered your password (**lsucs**), `apt` will tell you which packages it plans to install and how much extra disk space they'll take up. Press **Y** and hit **Enter** to continue, and the installation will proceed.

Set Global ServerName to Suppress Syntax Warnings

Next, we will add a single line to the `/etc/apache2/apache2.conf` file to suppress a warning message. While harmless, if you do not set `ServerName` globally, you will receive the following warning when checking your Apache configuration for syntax errors:

```
$ sudo apache2ctl configtest
```

```
Output
AH00558: apache2: Could not reliably determine the server's fully
qualified domain name, using 127.0.1.1. Set the 'ServerName' directive
globally to suppress this message
Syntax OK
```

Open up the main configuration file with your text edit:

```
$ sudo nano /etc/apache2/apache2.conf
```

Inside, at the end of the file, add a `ServerName` directive, pointing to your server IP address 10.15.14.64, or just 127.0.0.1 for our class testing.

```
ServerName 127.0.0.1
```

Save the file when you are finished (ctrl-O then Enter to accept the default filename), and then exit the editor (ctrl-X).

Next, check for syntax errors by typing:

```
$ sudo apache2ctl configtest
```

Since we added the global `ServerName` directive, all you should see now is:

```
Output
Syntax OK
```

Restart Apache to implement your changes:

```
$ sudo service apache2 restart
```

You can now begin adjusting the firewall.

Adjust the Firewall to Allow Web Traffic

Next, assuming that you have followed the initial server setup instructions to enable the UFW firewall, make sure that your firewall allows HTTP and HTTPS traffic. You can make sure that UFW has an application profile for Apache like so:

```
$ sudo ufw app list
```

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

If you look at the `Apache Full` profile, it should show that it enables traffic to ports 80 and 443:

```
$ sudo ufw app info "Apache Full"
```

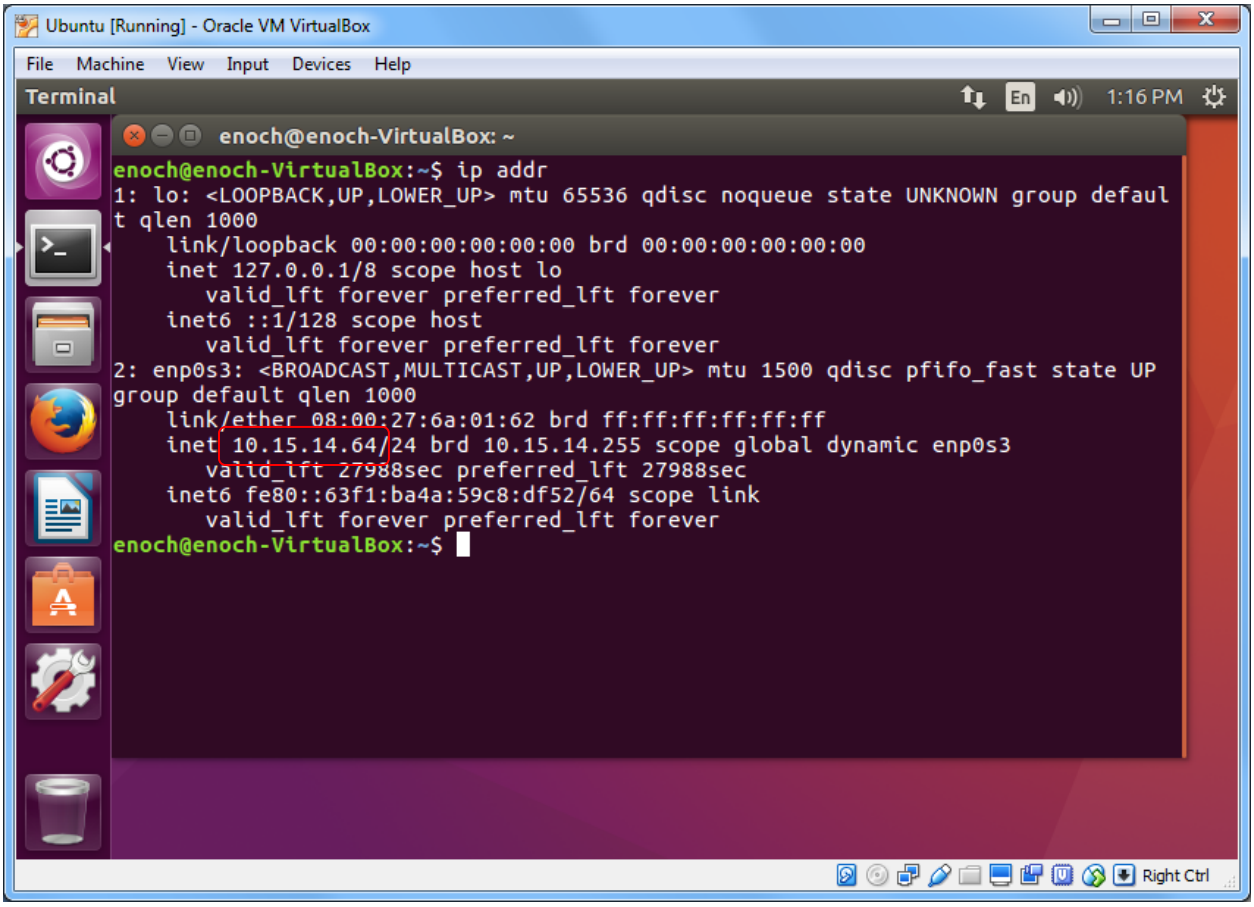
```
Output
Profile: Apache Full
Title: Web Server (HTTP,HTTPS)
Description: Apache v2 is the next generation of the omnipresent Apache
web server.

Ports:
  80,443/tcp
```

Allow incoming traffic for this profile:

```
$ sudo ufw allow in "Apache Full"
```

You can do a spot check right away to verify that everything went as planned by visiting your server's IP address in your web browser. To find out your server’s IP address type

```
$ ip addr
```



You'll see your IP address something like 10.15.x.x or 192.168.x.x

Use a browser from another computer. Type in the IP address that you got.

You will see the default Apache2 Ubuntu web page, which is there for informational and testing purposes. It should look something like this:

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

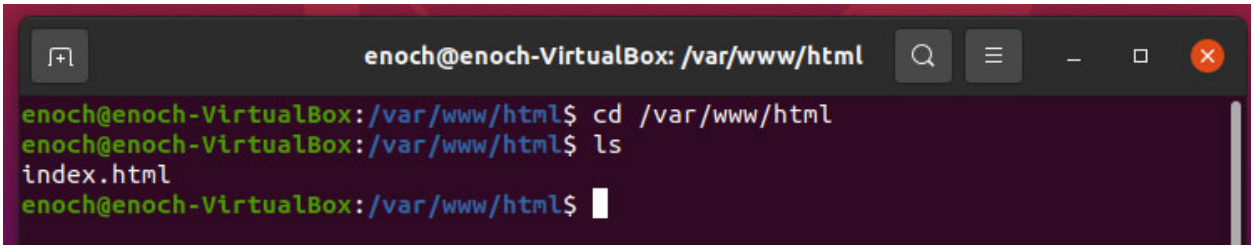
Please use the `ubuntu-bug` tool to report bugs in the Apache2 package with Ubuntu. However, check **existing bug reports** before reporting a new bug.

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

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

This default webpage is stored in the directory **/var/www/html** in the file named **index.html**. You can go to it by changing your current directory with the **cd** command and then list the directory content with ls

```
$ cd /var/www/html
$ ls
```



The webpage files are stored in the directory /var/www/html, and only the root user (with admin privileges) can add new files or modify existing ones. **nano** is a text editor. So to create a new file you need to add **sudo** in front of the **nano** command.

```
$sudo nano index.html
```

~~To make our file creation and editing a little easier, we will set read-write-execute privileges to everyone for all the files in that directory. (You do not want to do this in a real production server.)~~

~~From the Linux Terminal window prompt type the following two commands~~

```
$ sudo chmod 777 /var
$ sudo chmod 777 /var/www
$ sudo chmod 777 /var/www/html
```

~~then do~~

```
$ ls -l
```

~~to see the privilege changes made.~~

Step 3: Install MySQL

Now that you have the web server up and running, it is time to install MySQL. MySQL is a database management system. Basically, it will organize and provide access to databases where our site can store information.

Again, we can use `apt` to acquire and install our software. This time, we'll also install some other "helper" packages that will assist us in getting our components to communicate with each other:

```
$ sudo apt-get install mysql-server
```

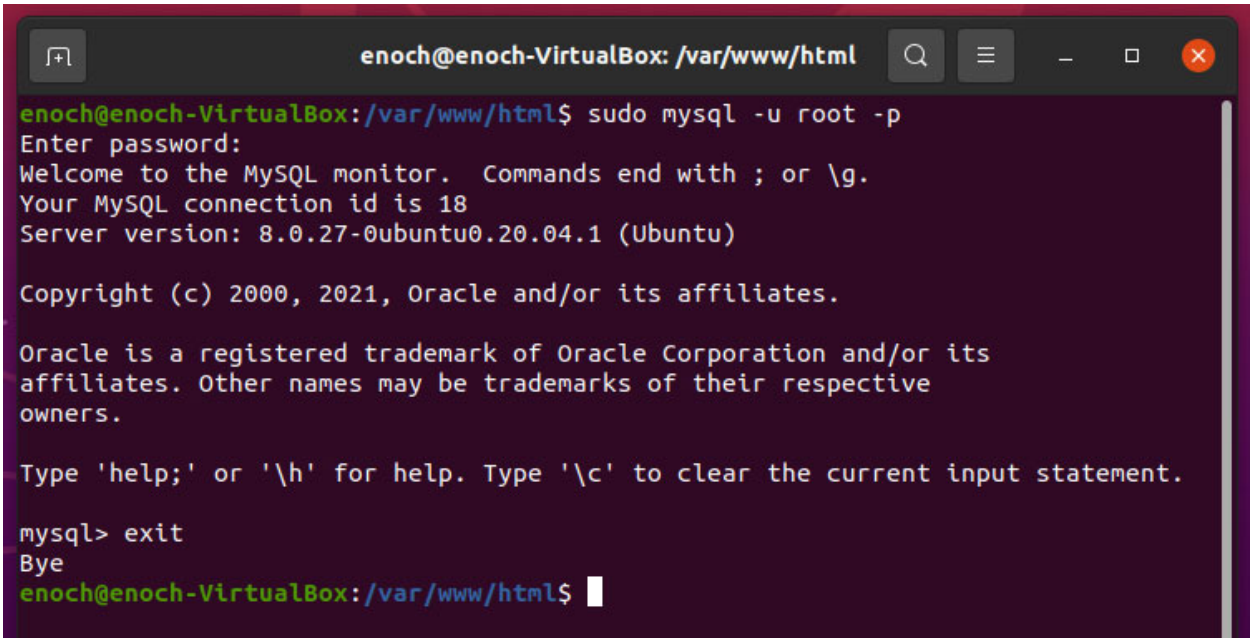
Again, you will be shown a list of the packages that will be installed, along with the amount of disk space they'll take up. Enter **Y** to continue.

~~During the installation, your server will ask you to select and confirm a password for the MySQL "root" user. This is an administrative account in MySQL that has increased privileges. Think of it as being similar to the root account for the server itself (the one you are configuring now is a MySQL-specific account, however). For our testing purposes, use **lsucs** for the root password.~~

At this point, your database system is now set up. Enter the following command to test your MySQL installation:

```
$ sudo mysql -u root -p
```

The password is your Ubuntu root password which should be **lsucs**. You should get the mysql> prompt.



Enter `exit` to exit MySQL.

~~Optional~~

~~No need to do this for our test system. Do this for a production system. When the installation is complete, we want to run a simple security script that will remove some dangerous defaults and lock down access to our database system a little bit. Start the interactive script by running:~~

```
$ mysql_secure_installation
```

~~You will be asked to enter the password you set for the MySQL root account. Next, you will be asked if you want to configure the `VALIDATE PASSWORD PLUGIN`. Answer **Y**.~~

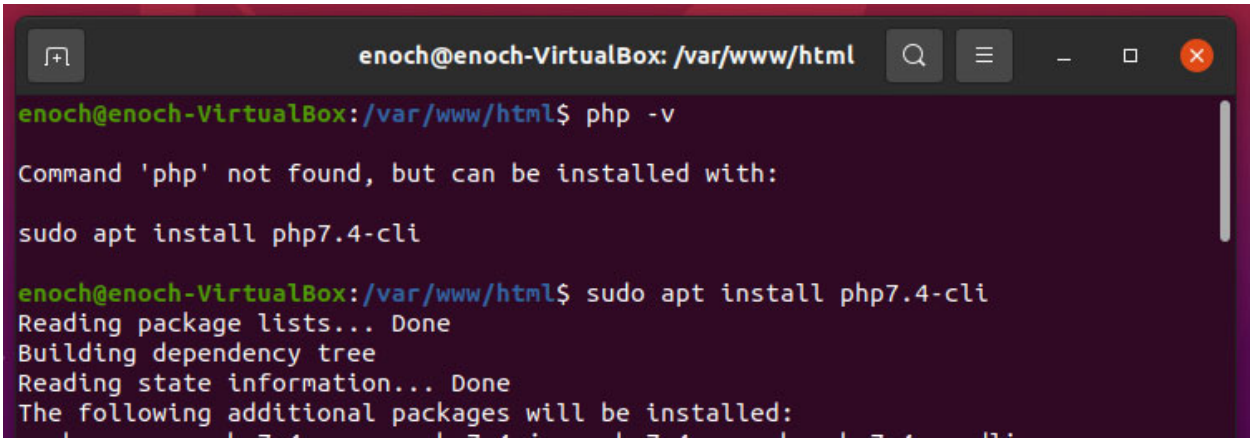
~~For the rest of the questions, you should press **Y** and hit the **Enter** key at each prompt. This will remove some anonymous users and the test database, disable remote root logins, and load these new rules so that MySQL immediately respects the changes we have made.~~

Step 4: Install PHP

PHP is the component of our setup that will process code to display dynamic content. It can run scripts, connect to our MySQL databases to get information, and hand the processed content over to our web server to display.

First type in the following command to find out the correct version of php to install. In the example below the current version is 7.4.

```
$ php -v

Command 'php' not found, but can be installed with:

sudo apt install php7.4-cli
```



Then type in the suggested command from above to install php.

```
$ sudo apt install php7.4-cli
```

Install the following two supporting libraries. Replace the version number with the correct current version

```
$ sudo apt-get install libapache2-mod-php7.4
$ sudo apt-get install php7.0-mcrypt
$ sudo apt-get install php7.4-mysql
```

(alternate way) Find the most current version of the libapache2-mod-php package to install

```
$ apt-cache search libapache2-mod-php
libapache2-mod-php7.0 - server-side, HTML-embedded scripting language
(Apache 2 module)
```

This should install PHP without any problems. We'll test this in a moment.

Optional Config the startup file

Currently, if a user requests a directory from the server, Apache will first look for a file called `index.html`. If we want to tell our web server to prefer PHP files, we can do the following to make Apache look for an `index.php` file first.

To do this, type this command to open the `dir.conf` file in a text editor with root privileges:

```
$ sudo nano /etc/apache2/mods-enabled/dir.conf
```

It will look like this:

```
<IfModule mod_dir.c>
    DirectoryIndex index.html index.cgi index.pl index.php index.xhtml
index.htm
</IfModule>
```

We want to move the PHP index file highlighted above to the first position after the `DirectoryIndex` specification, like this:

```
<IfModule mod_dir.c>
    DirectoryIndex index.php index.html index.cgi index.pl index.xhtml
index.htm
</IfModule>
```

When you are finished, save and close the file by pressing **Ctrl-X**. You'll have to confirm the save by typing **Y** and then hit **Enter** to confirm the file save location.

After this, we need to restart the Apache web server in order for our changes to be recognized. You can do this by typing this:

```
$ sudo service apache2 restart
```

We can also check on the status of the `apache2` service using `systemctl`:

```
$ sudo systemctl status apache2
```

```
Sample Output
● 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 Wed 2016-04-13 14:28:43 EDT; 45s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 13581 ExecStop=/etc/init.d/apache2 stop (code=exited,
status=0/SUCCESS)
  Process: 13605 ExecStart=/etc/init.d/apache2 start (code=exited,
status=0/SUCCESS)
    Tasks: 6 (limit: 512)
   CGroup: /system.slice/apache2.service
           ├─13623 /usr/sbin/apache2 -k start
           ├─13626 /usr/sbin/apache2 -k start
           ├─13627 /usr/sbin/apache2 -k start
           ├─13628 /usr/sbin/apache2 -k start
           ├─13629 /usr/sbin/apache2 -k start
           └─13630 /usr/sbin/apache2 -k start

Apr 13 14:28:42 ubuntu-16-lamp systemd[1]: Stopped LSB: Apache2 web
server.
Apr 13 14:28:42 ubuntu-16-lamp systemd[1]: Starting LSB: Apache2 web
server...
Apr 13 14:28:42 ubuntu-16-lamp apache2[13605]:  * Starting Apache httpd
web server apache2
Apr 13 14:28:42 ubuntu-16-lamp apache2[13605]: AH00558: apache2: Could not
reliably determine the server's fully qualified domain name, using
127.0.1.1. Set the 'ServerNam
Apr 13 14:28:43 ubuntu-16-lamp apache2[13605]:  *
Apr 13 14:28:43 ubuntu-16-lamp systemd[1]: Started LSB: Apache2 web
server.
```

Step 5: Test PHP Processing on your Web Server

In order to test that our system is configured properly for PHP, we can create a very basic PHP script.

We will call this script `info.php`. In order for Apache to find the file and serve it correctly, it must be saved to a very specific directory, which is called the "web root".

In Ubuntu, this directory is located at **`/var/www/html`**. We can create the file at that location by typing:

```
$ sudo nano /var/www/html/info.php
```

This will open a blank file. We want to put the following text, which is valid PHP code, inside the file:

```
<?php
phpinfo();
?>
```

When you are finished, save and close the file.

Now we can test whether our web server can correctly display content generated by a PHP script. To try this out, we just have to visit this page in our web browser. You'll need your server's IP address again. Again, to find your server IP address, type

```
$ ip addr
```

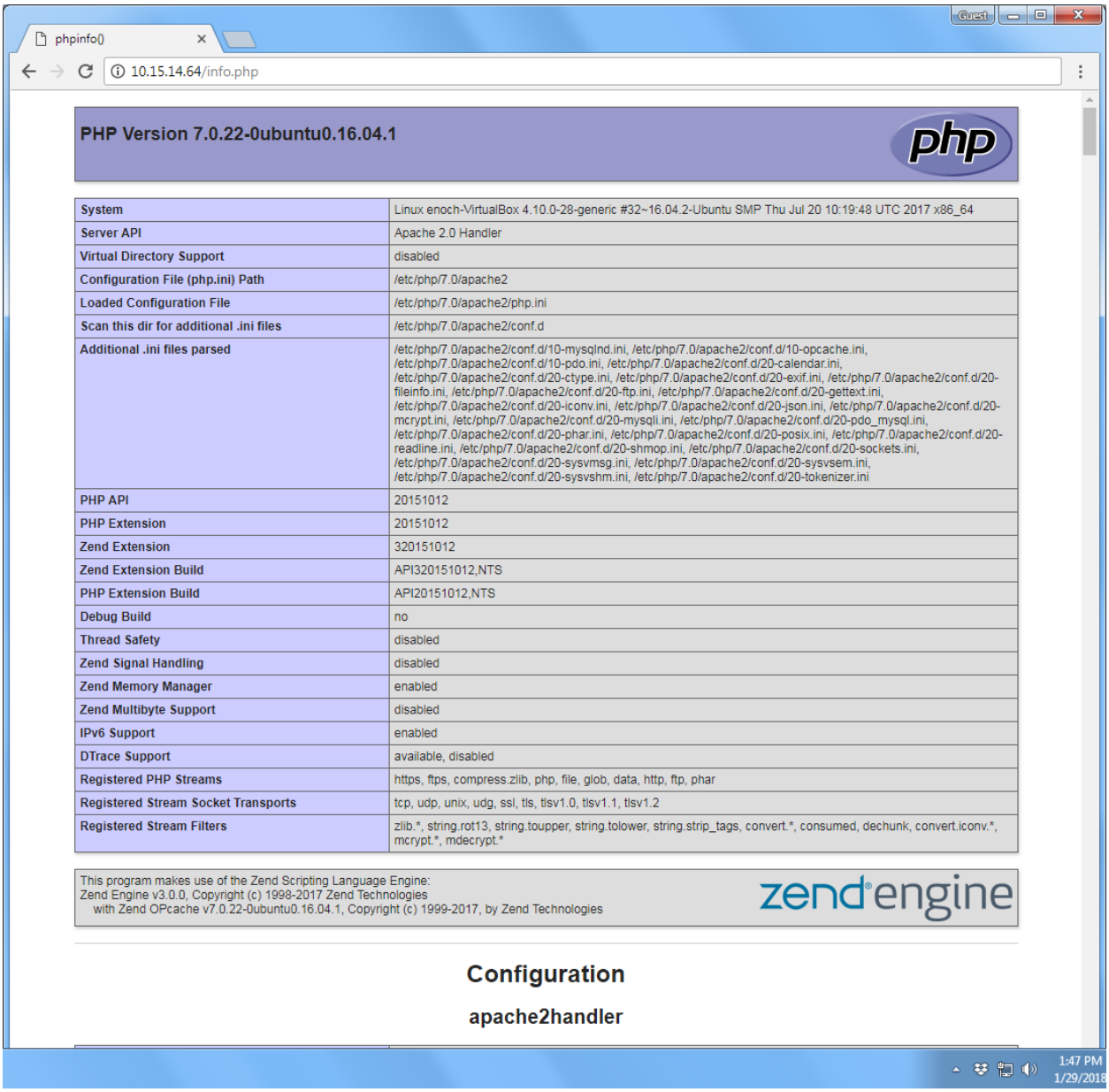
Using a browser on the same server computer, type in the address you want to visit:

```
http://your_server_IP_address/info.php
```

e.g.

```
http://10.15.15.169/info.php
```

Something similar to the following page should be displayed.



This page basically gives you 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 this was successful, then your PHP is working as expected.

You probably want to remove the file info.php after this test in a real production setup because it could actually give information about your server to unauthorized users. To do this, you can type this:

```
$ sudo rm /var/www/html/info.php
```

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

Conclusion

Now that you have a LAMP stack installed, you have many choices for what to do next. Basically, you've installed a platform that will allow you to install most kinds of websites and web software on your server.

Optional

1. In all of the above commands, you need to add `sudo` in front because those commands need to have admin privileges. You can issue the following command once and then you don't need to add `sudo` for all subsequent commands that need admin privileges.

   ```
   $ sudo -i
   ```

2. Filezilla allows you to upload files to your server from another computer. To install Filezilla, do

   ```
   $ sudo apt install filezilla
   ```

3. You should ensure that connections to your web server are secured, by serving them via HTTPS. The easiest option here is to use Let's Encrypt to secure your site with a free TLS/SSL certificate.