To Install Linux, Apache, MySQL, PHP (LAMP) Stack on Ubuntu

https://www.digitalocean.com/community/tutorials/how-to-install-lamp-stack-on-ubuntu

https://www.cherryservers.com/blog/install-lamp-on-ubuntu-22-04

https://cloudinfrastructureservices.co.uk/how-to-install-wordpress-on-ubuntu-20-04/

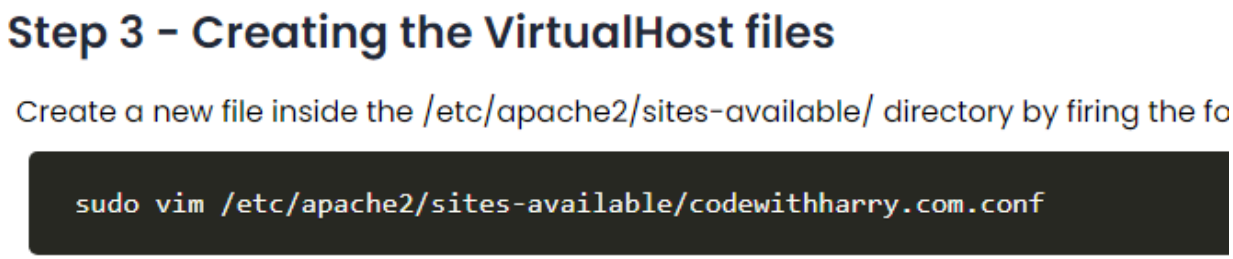
https://www.codewithharry.com/blogpost/install-phpmyadmin-ubuntu/

Step — Configure & creating multiple virtual host

https://www.codewithharry.com/blogpost/host-multiple-websites-ubuntu-vps/

sudo mkdir /var/www/your_domain

sudo chown -R $USER:$USER /var/www/your_domain



sudo nano /etc/apache2/sites-available/your_domain.conf

This will create a new blank file. Add in the following bare-bones configuration with your own domain name:

```
<VirtualHost *:80>
    ServerName your_domain
    ServerAlias www.your_domain
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE LOG DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Save and close the file when you're done. If you're using `nano`, do that by pressing `CTRL+X`, then `Y` and `ENTER`.

sudo a2ensite your_domain

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 override your virtual host. To disable Apache's default website, type:

sudo a2dissite 000-default

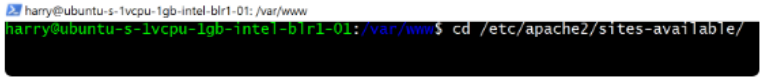
To make sure your configuration file doesn't contain syntax errors, run the following command:

sudo apache2ctl configtest

Step 4 - Enable the VirtualHosts

In order for these virtual host files to function correctly, we need to enable them.
Enter the directory where we have created virtual hosts:

```
cd /etc/apache2/sites-available/
```



Execute the following commands to enable the virtual hosts:

```
sudo a2ensite codewithharry.com.conf
sudo a2ensite programmingwithharry.com.conf
```

Finally, you will have to restart the apache server:

```
sudo service apache2 restart
```

Finally, reload Apache so these changes take effect:

sudo systemctl reload apache2

nano /var/www/your_domain/index.html

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

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

sudo systemctl reload apache2

Server Ip Pointing on Domain name

My Domains / Domain Settings

DNS Management

codewithharry.in

Records

Last updated 31-05-2021 17:46 PM

Type	Name	Value	TTL	
A	@	165.22.222.8	600 seconds	
A	www	165.22.222.8	600 seconds	
CNAME	_domainconnect	_domainconnect.gd.domaincontrol.com	1 Hour	

Step — Installing Apache and Updating the Firewall

sudo -i for changing root user
sudo chmod -R 777 /var/www/html

sudo apt update

sudo apt install apache2 –y

sudo systemctl status apache2

sudo apt update

sudo apt install apache2

or

sudo apt install apache2 -y

sudo systemctl status apache2

sudo ufw status //should be inactive

//if active

sudo ufw allow 80/tcp

sudo ufw reload

sudo ufw status

check in browser http://server-ip-address

```
sudo systemctl status apache2
```

```
cherry@ubuntu:~$
cherry@ubuntu:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-09-17 21:33:08 UTC; 3min 39s ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 2790 (apache2)
      Tasks: 55 (limit: 4675)
     Memory: 5.0M
        CPU: 62ms
     CGroup: /system.slice/apache2.service
             ├─2790 /usr/sbin/apache2 -k start
```

sudo ufw app list

Once the installation is finished, you'll need to adjust your firewall settings to allow HTTP traffic. Ubuntu's default firewall configuration tool is called Uncomplicated Firewall (UFW). It has different application profiles that you can leverage. To list all currently available UFW application profiles, execute this command:

```
$ sudo ufw app list
```

Copy

```
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 don't yet 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"
```

Copy

Verify the change with:

```
$ sudo ufw status
```

Copy

```
Output
Status: active

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

If you have a UFW firewall enabled, you need to adjust the firewall settings to allow HTTP traffic. UFW, short for Uncomplicated Firewall, is Ubuntu's default firewall configuration.

To allow HTTP traffic, you need to allow connections on port 80. To accomplish this, run the following command:

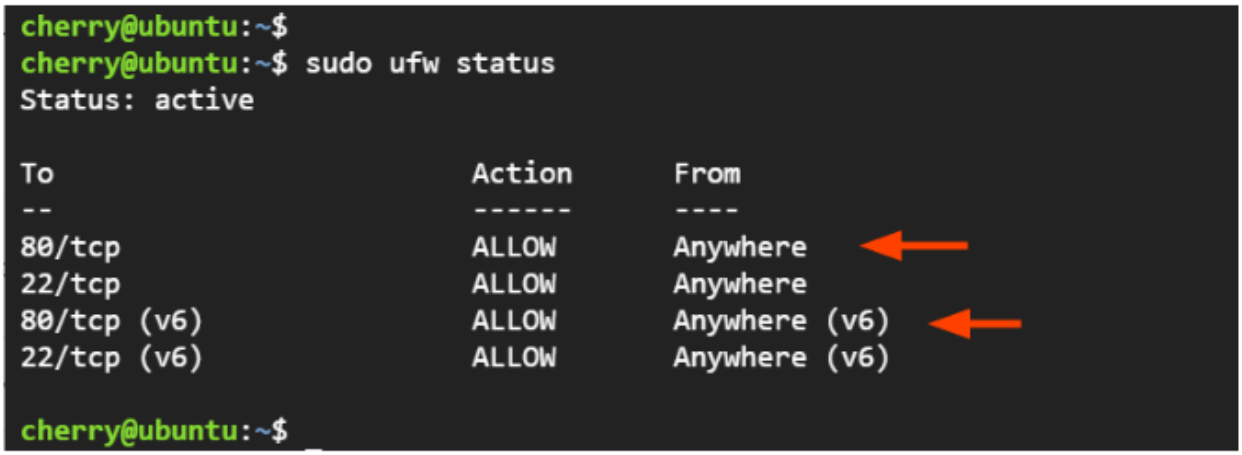
```
sudo ufw allow 80/tcp
```

To apply the changes, reload the firewall.

```
sudo ufw reload
```

To confirm the status of the firewall, run the following command:

```
sudo ufw status
```



From the output, we can see that port 80 is open on the firewall, and HTTP traffic is allowed. You can confirm this by visiting the server's IP address.

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 (view 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
```

The default Ubuntu Apache web page is there for informational and testing purposes. Below is an example of the Apache default web page for Ubuntu 22.04:

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 to 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 ens3 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'
```

Copy

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.

An alternative method is to use the `curl` utility to contact an outside party to tell you how it sees your server. This is done by asking a specific server what your IP address is:

```
$ curl http://icanhazip.com
```

Copy

Whichever method you choose, type in your IP address into your web browser to verify that your server is running.

Apache2 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 and is managed using systemd, so to start/stop the service use `systemctl start apache2` and `systemctl stop apache2`, and use `systemctl status apache2` and `journalctl -u apache2` to check status. `system` and `apache2ctl` can also be used for service management if desired. **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 outside 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.

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 their respective packages, not to the web server itself.

Step — Installing PHP & PHP MODULES

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

php –v

```
$ sudo apt install php-mysql php-cgi php-cli php-gd -
```

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.

Step 4:Install PHP and PHP modules

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

php --version

php -m //list of module for showing

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

http://server-ip/info.php

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

sudo apt install

ghostscript \

libapache2-mod-php \

php \

php-bcmath \

php-curl \

php-imagick \

php-intl \

php-json \

php-mbstring \

php-mysql \

php-xml \

php-zip

Changing Apache's Directory Index (Optional)

In some cases, you'll want to modify the way that Apache serves files when a directory is requested. Currently, if a user requests a directory from the server, Apache will first look for a file called `index.html`. We want to tell the web server to prefer PHP files over others, to make Apache look for an `index.php` file first. If you don't do that, an `index.html` file placed in the document root of the application will always take precedence over an `index.php` file.

To make this change, open the `dir.conf` configuration file in a text editor of your choice. Here, we'll use `nano`:

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

Copy

It will look like this:

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

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

Move the PHP index file (highlighted above) to the first position after the `DirectoryIndex` specification, like this:

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

```
<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`. Confirm the save by typing `Y` and then hit `ENTER` to verify the file save location.

After this, restart the Apache web server in order for your changes to be recognized. You can do that with the following command:

```
$ sudo systemctl restart apache2
```

Copy

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

```
$ sudo systemctl status apache2
```

Copy

The WordPress package we are going to install is the official release from **WordPress.org**. First we will make the directory and change its permissions to download our installation in.

sudo mkdir -p /srv/www

sudo chown www-data: /srv/www

```
sudo chown -R www-data:www-data /var/www/
```

The following instruction downloads the WordPress package.

curl https://wordpress.org/latest.tar.gz | sudo -u www-data tar zx -C /srv/www

```
php -m
```

```
cherry@ubuntu:~$
cherry@ubuntu:~$ php -m
[PHP Modules]
calendar
Core
ctype
date
exif
FFI
fileinfo
filter
ftp
gettext
hash
iconv
json
libxml
mbstring
```

Step — Installing Mariydb

Step 2: Install MariaDB database server

sudo apt install mariadb-server mariadb-client -y

mariadb --version

Step 2: Secure MariaDB database server

sudo mysql_secure_installation

```
cherry@ubuntu:~$
cherry@ubuntu:~$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):    Press ENTER to continue
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] n
 ... skipping.

You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
 ... Success!
```

```
By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] Y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] Y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] Y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] Y
 ... Success!
```

CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';

CREATE DATABASE mydatabase1;

CREATE USER 'rajivuser'@'%' IDENTIFIED BY 'root';

CREATE USER 'rajivuser'@'localhost' IDENTIFIED BY 'root';

For showing user list

SELECT User, Host FROM mysql.user;

GRANT ALL PRIVILEGES ON *.* TO 'rajivuser'@'localhost' WITH GRANT OPTION;

GRANT ALL ON example_database.* TO 'example_user'@'%';

FLUSH PRIVILEGES;

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';

```
MariaDB [(none)]> create user "wordpress"@"%" identified by "password";
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> grant all privileges on wordpress.* to "wordpress"@"%";
Query OK, 0 rows affected (0.01 sec)
```

sudo apt install phpmyadmin

Password Tiger@12345

sudo phpenmod mbstring
sudo phpenmod mysqli

sudo systemctl restart apache2

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';

https://askubuntu.com/questions/1417547/ubuntu-22-04-phpmyadmin-not-working

4

is the PHP code returned from the page and not executed ?

if yes, seems like you did not install libapache2-mod-php(version), where version is your PHP version, i.e for PHP 8.1:

> sudo apt install libapache2-mod-php8.1

after that, reload the apache :

> sudo systemctl restart apache2

and try to access `phpmyadmin`

hope this help

sudo mysql_secure_installation

```
$ sudo mysql_secure_installation
```

Copy

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:

```
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 doesn’t involve using a password, **even when one is set**, you should define a strong password here as an additional safety measure.

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:

```
mysql> CREATE USER 'example_user'@'%' IDENTIFIED BY 'password';
```

Copy

Note: The previous `ALTER USER` statement sets the **root** MySQL user to authenticate with the `caching_sha2_password` plugin. Per the official MySQL documentation, `caching_sha2_password` is MySQL’s preferred authentication plugin, as it provides more secure password encryption than the older, but still widely used, `mysql_native_password`.

However, some versions of PHP don’t work reliably with `caching_sha2_password`. PHP has reported that this issue was fixed as of PHP 7.4, but if you encounter an error when trying to log in to phpMyAdmin later on, you may want to set **root** to authenticate with `mysql_native_password` instead:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'p
```

Copy

```
l';
```

sudo mysql -u root

The above instructions opens MySQL's own terminal. We create a separate database titled 'wordpress-db' and create a user identified by a password of your choosing.

```
CREATE DATABASE wordpress-db;

CREATE USER wordpress-db@localhost IDENTIFIED BY 'password';
```

We grant it privileges and flush them.

```
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER

FLUSH PRIVILEGES;

exit
```

To connect the database, we copy the config file and set the configuration in the file. Replace with your database name, db username and password.

```
sudo -u www-data cp /srv/www/wordpress/wp-config-sample.php /srv/www/wordpress/wp-config.php

sudo -u www-data sed -i 's/database_name_here/wordpress/' /srv/www/wordpress/wp-config.php

sudo -u www-data sed -i 's/username_here/wordpress/' /srv/www/wordpress/wp-config.php

sudo -u www-data sed -i 's/password_here//' /srv/www/wordpress/wp-config.php
```

Open the following file using nano.

```
sudo -u www-data nano /srv/www/wordpress/wp-config.php
```

Installation & Configure Of PHPMADMIN

sudo apt install phpmyadmin

3. **Configure phpMyAdmin**: During the installation process, you'll be prompted to configure phpMyAdmin. Here are the steps you'll typically encounter:
   - When asked to choose the web server, you'll likely see options like Apache and Lighttpd. Select the web server you're using (e.g., Apache) by pressing the spacebar, and then hit `Enter`.
   - You'll be asked if you want to configure the database for phpMyAdmin with `dbconfig-common`. Select `Yes`.
   - Provide the MySQL or MariaDB administrator password when prompted.
4. **Enable PHP Extensions**: After installing phpMyAdmin, you may need to enable the PHP extensions it requires. Run the following commands to enable the necessary extensions:

```
Copy code

sudo phpenmod mbstring
sudo phpenmod mysqli
```

5. **Restart Apache**: After installing and configuring phpMyAdmin, restart the Apache web server to apply the changes:

```
Copy code

sudo systemctl restart apache2
```

6. **Access phpMyAdmin**: Once everything is set up, you can access phpMyAdmin by navigating to `http://your_server_ip/phpmyadmin` in your web browser. Replace `your_server_ip` with the public IP address or domain name of your server.
7. **Login to phpMyAdmin**: You'll be prompted to log in with your MySQL or MariaDB username and password. After logging in, you'll have acc to the phpMyAdmin dashboard to manage your databases.

Configure Apache & VITUAL HOST [OPTIONAL]

Now, we need to configure the Apache web server site for WordPress. We create a file in the with the path **/etc/apache2/sites-available/wordpress.conf** and copy the following into the conf file (edit your ServerName and ServerAlias).

```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    DocumentRoot /srv/www/wordpress
    ServerName sitename.com
    ServerAlias www.sitename.com
    <Directory /srv/www/wordpress/>
        Options FollowSymlinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/wordpress_error.log
    CustomLog ${APACHE_LOG_DIR}/wordpress_access.log combined
</VirtualHost>
```

We will now enable the site and enable URL writing.

```
sudo a2ensite wordpress
sudo a2enmod rewrite
```

Now, we reload apache2.

```
sudo service apache2 reload
```

Run the following command to disable the default page so that WordPress loads.

```
sudo a2dissite 000-default
```

Apache web server provides a modular design that allows you to host multiple domains or websites using virtual hosts. Virtual hosting is a concept that allows you to host multiple websites on a single server. A virtual host file is a configuration file that defines the domain to be hosted on the server. It also defines the path of the domain's website files and other crucial parameters.

In this section, we will configure an Apache virtual for our domain.

To get started, create a website directory for your domain in the `/var/www/` path.

```
sudo mkdir -p /var/www/domain.com
```

Next, assign the following directory ownership.

```
sudo chown -R $USER:$USER /var/www/domain.com
```

The `$USER` environment specifies the currently logged-in user. This implies that the website directory will be owned by the logged-in user and not by root.

Next, assign directory permissions.

```
sudo chmod -R 755 /var/www/domain.com
```

Moving on, create a sample html file inside the website directory. This will be used to prove that the virtual host is working. So, create an `index.html` file using your preferred text editor. In this case, we are using nano editor.

```
sudo nano /var/www/domain.com/index.html
```

Paste the following lines of code.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Welcome to our sample domain!</title>
    </head>
    <body>
        <h1>Success!  The sample domain virtual host is working!</h1>
    </body>
</html>
```

Save and exit. Next, create a virtual host configuration file in the `/etc/apache2/sites-available/` directory.

```
sudo nano /etc/apache2/sites-available/domain.com.conf
```

Paste the following lines of code and save the changes.

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

Once done, save and close the file. Next, enable the virtual host site using the `a2ensite` tool.

```
sudo a2ensite domain.com.conf
```

Next, disable the default site, `000-default.conf` , using the `a2dissite` command line tool.

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

To ensure that the Apache configuration is correct, run the command:

```
sudo apache2ctl configtest
```


```
sudo apache2ctl configtest
```


```
cherry@ubuntu:~$
cherry@ubuntu:~$
cherry@ubuntu:~$ sudo apache2ctl configtest
Syntax OK
cherry@ubuntu:~$
cherry@ubuntu:~$
cherry@ubuntu:~$
```

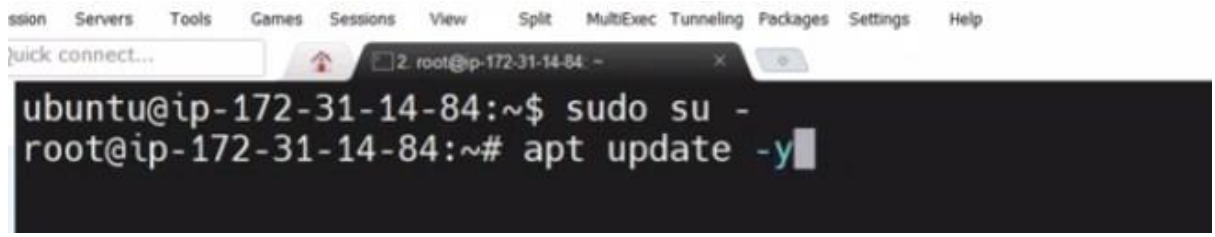
Finally, restart Apache to apply all the changes made.

```
sudo systemctl restart apache2
```

Now verify that your domain name is being served by Apache by visiting your domain's URL

```
http://www.domain.com
```

Wordpress Installation on AWS




```
root@ip-172-31-14-84:~# apt upgrade -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

```
root@ip-172-31-14-84:~#
root@ip-172-31-14-84:~# apt install apache2 -y
Reading package lists... Done
```

```
root@ip-172-31-14-84:~# systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor
     Active: active (running) since Sun 2022-10-23 03:40:37 UTC; 28s ago
       Docs: https://httpd.apache.org/docs/2.4/
   Main PID: 14153 (apache2)
      Tasks: 55 (limit: 1143)
     Memory: 5.0M
        CPU: 33ms
     CGroup: /system.slice/apache2.service
             ├─14153 /usr/sbin/apache2 -k start
             ├─14155 /usr/sbin/apache2 -k start
             └─14156 /usr/sbin/apache2 -k start
```

```
root@ip-172-31-14-84:~# systemctl enable apache2
Synchronizing state of apache2.service with SysV service scr
Executing: /lib/systemd/systemd-sysv-install enable apache2
```

```
root@ip-172-31-14-84:~# apt install mariadb-server mariadb-client -y
```

```
root@ip-172-31-14-84:~# systemctl start mariadb
root@ip-172-31-14-84:~# systemctl status mariadb
```

```
ubuntu@ip-172-31-14-84:~$ sudo su -
root@ip-172-31-14-84:~# mysql_secure_installation
```

```
Change the root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
 ... Success!


By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] Y
```

```
Disallow root login remotely? [Y/n] Y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] Y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] Y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

```
Thanks for using MariaDB!
root@ip-172-31-14-84:~# systemctl restart mariadb
root@ip-172-31-14-84:~# apt install php php-mysql php-gd php-cli php-common -y
```

```
root@ip-172-31-14-84:~# apt install wget unzip -y
```

```
root@ip-172-31-14-84:~# ls
latest.zip  snap
root@ip-172-31-14-84:~# unzip latest.zip
```

```
root@ip-172-31-14-84:~# ls
latest.zip  snap  wordpress
root@ip-172-31-14-84:~# cd wordpress/
root@ip-172-31-14-84:~/wordpress# ls
index.php         wp-admin             wp-content         wp-load.ph
license.txt       wp-blog-header.php   wp-cron.php        wp-login.p
readme.html       wp-comments-post.php wp-includes        wp-mail.ph
wp-activate.php   wp-config-sample.php wp-links-opml.php  wp-setting
root@ip-172-31-14-84:~/wordpress# cd ..
root@ip-172-31-14-84:~# ls
latest.zip  snap  wordpress
root@ip-172-31-14-84:~# pwd
/root
root@ip-172-31-14-84:~# cp -r wordpress/* /var/www/html/
root@ip-172-31-14-84:~# cd /var/www/html/
root@ip-172-31-14-84:/var/www/html# ls -
```

```
root@ip-172-31-14-84:/var/www/html# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.6.7-MariaDB-2ubuntu1.1 Ubuntu 22.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> create database wordpress;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> create user "wpadmin" identified by "wpadminpass";
Query OK, 0 rows affected (0.003 sec)

MariaDB [(none)]> grant all privileges on wordpress.* to "wpadmin";
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]>
```

Wordpres installation on AWS

1. **Create an AWS Account:** If you don't already have an AWS account, sign up for one at https://aws.amazon.com/. You will need to provide billing information, but you can use the AWS Free Tier to get started without incurring charges for the first 12 months.
2. **Launch an EC2 Instance:**
    - Sign in to your AWS Management Console.
    - Navigate to the EC2 Dashboard.
    - Click the "Launch Instance" button to create a new virtual server.
    - Choose an Amazon Machine Image (AMI) based on your requirements. You can use an Amazon Linux AMI, which is suitable for most purposes.
    - Select an instance type, which determines the server's hardware resources. The t2.micro instance is part of the Free Tier.
    - Configure instance details, such as the number of instances, network settings, and storage.
    - Add storage for your instance. You can use the default settings or adjust the storage size as needed.
    - Configure security groups to control incoming traffic to your instance. At a minimum, allow SSH (port 22) and HTTP/HTTPS (ports 80 and 443).
    - Review and launch the instance, and create a new key pair or use an existing one to securely connect to your instance.
3. **Connect to Your EC2 Instance:**
    - Once the instance is running, use SSH to connect to it. Use the private key from the key pair you selected during instance launch.
    - The command to connect will look like this:

```css
ssh -i /path/to/your-key.pem ec2-user@your-instance-ip
```

4. **Install LAMP Stack:**
   - After connecting to your EC2 instance, you can install the LAMP (Linux, Apache, MySQL, PHP) stack to host WordPress.
   - Run the following commands to install Apache, MySQL, and PHP:

```bash
sudo yum install httpd -y
sudo systemctl start httpd
sudo systemctl enable httpd
sudo yum install mysql-server -y
sudo systemctl start mysqld
sudo systemctl enable mysqld
sudo yum install php php-mysql -y
```

5. **Secure MySQL:**
   - Run the MySQL secure installation script to set a root password and secure your MySQL installation.

```
sudo mysql_secure_installation
```

6. **Download and Configure WordPress:**
   - Download the latest WordPress release and extract it in the Apache web server's document root:

```bash
cd /var/www/html
sudo wget https://wordpress.org/latest.tar.gz
sudo tar -xzf latest.tar.gz
```

6. **Download and Configure WordPress:**
   - Download the latest WordPress release and extract it in the Apache web server's document root:

```bash
cd /var/www/html
sudo wget https://wordpress.org/latest.tar.gz
sudo tar -xzf latest.tar.gz
```

7. **Create a MySQL Database for WordPress:**
   - Log in to MySQL and create a new database and user for WordPress.

```css
mysql -u root -p
```

   Enter your MySQL root password, then run the following SQL commands:

```sql
CREATE DATABASE wordpress;
CREATE USER 'wordpressuser'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON wordpress.* TO 'wordpressuser'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

8. **Configure WordPress:**
   - Rename the `wp-config-sample.php` file to `wp-config.php` and edit it to provide your database details:

```arduino
cd /var/www/html/wordpress
sudo mv wp-config-sample.php wp-config.php
sudo nano wp-config.php
```

   Update the database name, user, and password you created earlier.

9. **Access WordPress Installation:**
    - Open a web browser and enter your instance's public IP address or domain name. You should see the WordPress installation wizard.
    - Complete the setup by providing your site title, username, password, and email address.
10. **Complete WordPress Setup:**
    - After the setup, you can log in to your WordPress admin dashboard and start customizing your site.
11. **Additional Configuration:**
    - You may want to configure your domain name, set up SSL/TLS certificates, and enhance security. AWS offers services like Route 53 and ACM for domain management and SSL certificates.
12. **Regular Backups and Updates:**
    - Ensure you regularly back up your WordPress site and keep both your WordPress and server software up to date to maintain security and performance.

This is a basic guide to installing WordPress on AWS. Depending on your specific requirements, you may need to configure additional services, such as an S3 bucket for media storage or CloudFront for content delivery. Always refer to the AWS documentation for the most up-to-date information and best practices.

Reg

LAMP Setup

https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-20-04

```
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"
```
Copy

You can verify the change with:

```
$ sudo ufw status
```
Copy

```
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:

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 $2; }' | sed 's/\/.*$//'
```
Copy

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.

An alternative method is to use the `curl` utility to contact an outside party to tell you how *it* sees your server. This is done by asking a specific server what your IP address is:

```
$ curl http://icanhazip.com
```
Copy

Regardless of the method you use to get your IP address, type it into your web browser's address bar to view the default Apache page.

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
```

Copy

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
```

Copy

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
```

Copy

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:

```
Output
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
```

Copy

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.

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
```

Copy

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

```
$ php -v
```

Copy

```
Output
PHP 7.4.3 (cli) (built: Jul  5 2021 15:13:35) ( 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

When using the Apache web server, you can create *virtual hosts* (similar to server blocks in Nginx) to encapsulate configuration details and host more than one domain from a single server. In this guide, we'll set up a domain called **your_domain**, but you should **replace this with your own domain name**.

> **Note**: In case you are using DigitalOcean as DNS hosting provider, you can check our product docs for detailed instructions on how to set up a new domain name and point it to your server.

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 **your_domain** 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 **your_domain** as follows:

```
$ sudo mkdir /var/www/your_domain
```
Copy

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/your_domain
```
Copy

Then, open a new configuration file in Apache's `sites-available` directory using your preferred command-line editor. Here, we'll use `nano`:

```
$ sudo nano /etc/apache2/sites-available/your_domain.conf
```
Copy

This will create a new blank file. Paste in the following bare-bones configuration:

/etc/apache2/sites-available/your_domain.conf

```
<VirtualHost *:80>
    ServerName your_domain
    ServerAlias www.your_domain
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/your_domain
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Save and close the file when you're done. If you're using `nano`, you can do that by pressing `CTRL+X`, then `Y` and `ENTER`.

With this `VirtualHost` configuration, we're telling Apache to serve `your_domain` using `/var/www/your_domain` 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.

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

```
$ sudo a2ensite your_domain
```
Copy

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
```
Copy

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

```
$ sudo apache2ctl configtest
```
Copy

Finally, reload Apache so these changes take effect:

```
$ sudo systemctl reload apache2
```
Copy

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

```
$ nano /var/www/your_domain/index.html
```
Copy

Include the following content in this file:

/var/www/your_domain/index.html

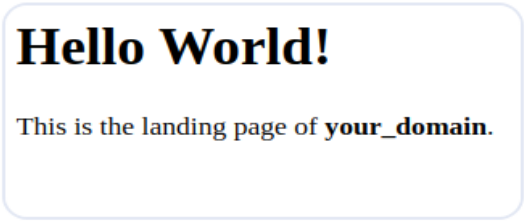
```
<html>
  <head>
    <title>your_domain website</title>
  </head>
  <body>
    <h1>Hello World!</h1>

    <p>This is the landing page of <strong>your_domain</strong>.</p>
  </body>
</html>
```

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

```
http://server_domain_or_IP
```

You'll see a page like this:



If you see this page, 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 nano /etc/apache2/mods-enabled/dir.conf
```
Copy

/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
```
Copy

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

Step 5 – Testing PHP Processing on your Web 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:

```
$ nano /var/www/your_domain/info.php
```

Copy

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

/var/www/your_domain/info.php

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

Copy

When you are finished, save and close the file.

To test this script, go to your web browser and access your server's domain name or IP address, followed by the script name, which in this case is `info.php`:

```
http://server_domain_or_IP/info.php
```

You'll see a page similar to this:

Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.3, Copyright (c), by Zend Technologies

zend engine

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/your_domain/info.php
```

Copy

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

To create a new database, run the following command from your MySQL console:

```
mysql> CREATE DATABASE example_database;
```

Copy

Now you can create a new user and grant them full privileges on the custom database you've just created.

The following command creates a new user named `example_user`, using `mysql_native_password` as default authentication method. We're defining this user's password as `password`, but you should replace this value with a secure password of your own choosing.

```
mysql> CREATE USER 'example_user'@'%' IDENTIFIED WITH mysql_native_password BY 'password';
```

Copy

Now we need to give this user permission over the `example_database` database:

```
mysql> GRANT ALL ON example_database.* TO 'example_user'@'%';
```

Copy

This will give the **example_user** user full privileges over the **example_database** database, while preventing this user from creating or modifying other databases on your server.

Now exit the MySQL shell with:

```
mysql> exit
```

Copy

You can test if the new user has the proper permissions by logging in to the MySQL console again, this time using the custom user credentials:

```
$ mysql -u example_user -p
```

Copy

Notice the `-p` flag in this command, which will prompt you for the password used when creating the **example_user** user. After logging in to the MySQL console, confirm that you have access to the **example_database** database:

```
mysql> SHOW DATABASES;
```

Copy

This will give you the following output:

```
Output
+--------------------+
| Database           |
+--------------------+
| example_database   |
| information_schema |
+--------------------+
```

Step 1 – Creating a MySQL Database and User for WordPress

The first step that we will take is a preparatory one. WordPress uses MySQL to manage and store site and user information. We have MySQL installed already, but we need to make a database and a user for WordPress to use.

To get started, log into the MySQL root (administrative) account by issuing this command (note that this is not the root user of your server):

```
$ mysql -u root -p
```
Copy

You will be prompted for the password you set for the MySQL root account when you installed the software.

Note: If you cannot access your MySQL database via root, as a `sudo` user you can update your root user's password by logging into the database like so:

```
$ sudo mysql -u root
```
Copy

Once you receive the MySQL prompt, you can update the root user's password. Here, replace `new_password` with a strong password of your choosing.

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'new_p
```
Copy

You may now type `EXIT;` and can log back into the database via password with the following command:

```
$ mysql -u root -p
```
Copy

Within the database, we can create an exclusive database for WordPress to control. You can call this whatever you would like, but we will be using the name **wordpress** in this guide. Create the database for WordPress by typing:

```
mysql> CREATE DATABASE wordpress DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```
Copy

Note: Every MySQL statement must end in a semi-colon (`;`). Check to make sure this is present if you are running into any issues.

Next, we are going to create a separate MySQL user account that we will use exclusively to operate our new database. Creating specific databases and accounts can support us from a management and security standpoint. We will use the name **wordpressuser** in this guide, but feel free to use whatever name is relevant for you.

We are going to create this account, set a password, and grant access to the database we created. We can do this by typing the following command. Remember to choose a strong password here for your database user where we have `password`:

```
mysql> CREATE USER 'wordpressuser'@'%' IDENTIFIED WITH mysql_native_password BY 'passwor
```
Copy

Next, let the database know that our **wordpressuser** should have complete access to the database we set up:

```
mysql> GRANT ALL ON wordpress.* TO 'wordpressuser'@'%';
```
Copy

You now have a database and user account, each made specifically for WordPress. We need to flush the privileges so that the current instance of MySQL knows about the recent changes we've made:

```
mysql> FLUSH PRIVILEGES;
```
Copy

Exit out of MySQL by typing:

```
mysql> EXIT;
```
Copy

In the next step, we'll lay some foundations for WordPress plugins by downloading PHP extensions for our server.

Step 2 – Installing Additional PHP Extensions

When setting up our LAMP stack, we only required a very minimal set of extensions in order to get PHP to communicate with MySQL. WordPress and many of its plugins leverage additional PHP extensions.

We can download and install some of the most popular PHP extensions for use with WordPress by typing:

```
$ sudo apt update
$ sudo apt install php-curl php-gd php-mbstring php-xml php-xmlrpc php-soap php-intl php-zip
```
Copy

This will lay the groundwork for installing additional plugins into our WordPress site.

Note: Each WordPress plugin has its own set of requirements. Some may require additional PHP packages to be installed. Check your plugin documentation to discover its PHP requirements. If they are available, they can be installed with `apt` as demonstrated above.

We will need to restart Apache to load these new extensions, we'll be doing more configurations on Apache in the next section, so you can wait until then, or restart now to complete the PHP extension process.

```
$ sudo systemctl restart apache2
```
Copy

Step 3 – Adjusting Apache's Configuration to Allow for .htaccess Overrides and Rewrites

Next, we will be making a few minor adjustments to our Apache configuration. Based on the prerequisite tutorials, you should have a configuration file for your site in the `/etc/apache2/sites-available/` directory.

In this guide, we'll use `/etc/apache2/sites-available/wordpress.conf` as an example here, but you should substitute the path to your configuration file where appropriate. Additionally, we will use `/var/www/wordpress` as the root directory of our WordPress install. You should use the web root specified in your own configuration. If you followed our LAMP tutorial, it may be your domain name instead of `wordpress` in both of these instances.

Note: It's possible you are using the `000-default.conf` default configuration (with `/var/www/html` as your web root). This is fine to use if you're only going to host one website on this server. If not, it's better to split the necessary configuration into logical chunks, one file per site.

With our paths identified, we can move onto working with `.htaccess` so that Apache can handle configuration changes on a per-directory basis.

Enabling .htaccess Overrides

Currently, the use of `.htaccess` files is disabled. WordPress and many WordPress plugins use these files extensively for in-directory tweaks to the web server's behavior.

Open the Apache configuration file for your website with a text editor like nano.

```
$ sudo nano /etc/apache2/sites-available/wordpress.conf
```
Copy

To allow `.htaccess` files, we need to set the `AllowOverride` directive within a `Directory` block pointing to our document root. Add the following block of text inside the `VirtualHost` block in your configuration file, making sure to use the correct web root directory:

/etc/apache2/sites-available/wordpress.conf

```
<Directory /var/www/wordpress/>
        AllowOverride All
</Directory>
```

When you are finished, save and close the file. In nano, you can do this by pressing `CTRL` and `X` together, then `Y`, then `ENTER`.

Enabling the Rewrite Module

Next, we can enable `mod_rewrite` so that we can utilize the WordPress permalink feature:

```
$ sudo a2enmod rewrite
```
Copy

This allows you to have more human-readable permalinks to your posts, like the following two examples:

```
http://example.com/2012/post-name/
http://example.com/2012/12/30/post-name
```

The `a2enmod` command calls a script that enables the specified module within the Apache configuration.

Enabling the Changes

Before we implement the changes we've made, check to make sure we haven't made any syntax errors by running the following test.

```
$ sudo apache2ctl configtest
```
Copy

You may receive output like the following:

```
Output
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127
Syntax OK
```

If you wish to suppress the top line, just add a `ServerName` directive to your main (global) Apache configuration file at `/etc/apache2/apache2.conf`. The `ServerName` can be your server's domain or IP address. This is just a message, however, and doesn't affect the functionality of your site. As long as the output contains `Syntax OK`, you are ready to continue.

Restart Apache to implement the changes. Make sure to restart now even if you have restarted earlier in this tutorial.

```
$ sudo systemctl restart apache2
```
Copy

Next, we will download and set up WordPress itself.

Step 4 – Downloading WordPress

Now that our server software is configured, we can download and set up WordPress. For security reasons in particular, it is always recommended to get the latest version of WordPress from their site.

Change into a writable directory (we recommend a temporary one like `/tmp`) and download the compressed release.

```
$ cd /tmp
$ curl -O https://wordpress.org/latest.tar.gz
```
Copy

Extract the compressed file to create the WordPress directory structure:

```
$ tar xzvf latest.tar.gz
```
Copy

We will be moving these files into our document root momentarily. Before we do, we can add a dummy `.htaccess` file so that this will be available for WordPress to use later.

Create the file by typing:

```
$ touch /tmp/wordpress/.htaccess
```
Copy

We'll also copy over the sample configuration file to the filename that WordPress reads:

```
$ cp /tmp/wordpress/wp-config-sample.php /tmp/wordpress/wp-config.php
```
Copy

We can also create the `upgrade` directory, so that WordPress won't run into permissions issues when trying to do this on its own following an update to its software:

```
$ mkdir /tmp/wordpress/wp-content/upgrade
```
Copy

Now, we can copy the entire contents of the directory into our document root. We are using a dot at the end of our source directory to indicate that everything within the directory should be copied, including hidden files (like the `.htaccess` file we created):

```
$ sudo cp -a /tmp/wordpress/. /var/www/wordpress
```
Copy

Ensure that you replace the `/var/www/wordpress` directory with the directory you have set up on your server.

Step 5 – Configuring the WordPress Directory

Before we do the web-based WordPress setup, we need to adjust some items in our WordPress directory.

Adjusting the Ownership and Permissions

An important step that we need to accomplish is setting up reasonable file permissions and ownership.

We'll start by giving ownership of all the files to the **www-data** user and group. This is the user that the Apache web server runs as, and Apache will need to be able to read and write WordPress files in order to serve the website and perform automatic updates.

Update the ownership with the `chown` command which allows you to modify file ownership. Be sure to point to your server's relevant directory.

```
$ sudo chown -R www-data:www-data /var/www/wordpress
```
Copy

Next we'll run two `find` commands to set the correct permissions on the WordPress directories and files:

```
$ sudo find /var/www/wordpress/ -type d -exec chmod 750 {} \;
$ sudo find /var/www/wordpress/ -type f -exec chmod 640 {} \;
```
Copy

These permissions should get you working effectively with WordPress, but note that some plugins and procedures may require additional tweaks.

Setting Up the WordPress Configuration File

Now, we need to make some changes to the main WordPress configuration file.

When we open the file, our first task will be to adjust some secret keys to provide a level of security for our installation. WordPress provides a secure generator for these values so that you do not have to try to come up with good values on your own. These are only used internally, so it won't hurt usability to have complex, secure values here.

To grab secure values from the WordPress secret key generator, type:

```
$ curl -s https://api.wordpress.org/secret-key/1.1/salt/
```
Copy

You will get back unique values that resemble output similar to the block below.

Warning! It is important that you request unique values each time. Do **NOT** copy the values below!

```
Output
define('AUTH_KEY',         '1jl/vqfs<XhdXoAPz9 DO NOT COPY THESE VALUES c_j{iwqD^<+c9.k<J@4H');
define('SECURE_AUTH_KEY',  'E2N-h2]Dcvp+aS/p7X DO NOT COPY THESE VALUES {Ka(f;rv?Pxf})CgLi-3');
define('LOGGED_IN_KEY',    'W(50,{W^,OPB%PB<JF DO NOT COPY THESE VALUES 2;y&,2m%3]R6DUth[;88');
define('NONCE_KEY',        'll,4UC)7ua+8<!4VM+ DO NOT COPY THESE VALUES #`DXF+[$atzM7 o^-C7g');
define('AUTH_SALT',        'koMrurzOA+|L_lG}kf DO NOT COPY THESE VALUES  07VC*Lj*lD&?3w!BT#-');
define('SECURE_AUTH_SALT', 'p32*p,]z%LZ+pAu:VY DO NOT COPY THESE VALUES C-?y+K0DK_+F|0h{!_xY');
define('LOGGED_IN_SALT',   'i^/G2W7!-1H2OQ+t$3 DO NOT COPY THESE VALUES t6**bRVFSD[Hi])-qS`|');
define('NONCE_SALT',       'Q6]U:K?j4L%Z]}h^q7 DO NOT COPY THESE VALUES 1% ^qUswWgn+6&xqHN&%');
```

These are configuration lines that we can paste directly in our configuration file to set secure keys.

These are configuration lines that we can paste directly in our configuration file to set secure keys. Copy the output you received now.

Next, open the WordPress configuration file:

```
$ sudo nano /var/www/wordpress/wp-config.php
```
Copy

Find the section that contains the example values for those settings.

/var/www/wordpress/wp-config.php

```
. . .

define('AUTH_KEY',         'put your unique phrase here');
define('SECURE_AUTH_KEY',  'put your unique phrase here');
define('LOGGED_IN_KEY',    'put your unique phrase here');
define('NONCE_KEY',        'put your unique phrase here');
define('AUTH_SALT',        'put your unique phrase here');
define('SECURE_AUTH_SALT', 'put your unique phrase here');
define('LOGGED_IN_SALT',   'put your unique phrase here');
define('NONCE_SALT',       'put your unique phrase here');

. . .
```

Delete those lines and paste in the values you copied from the command line:

/var/www/wordpress/wp-config.php

```
. . .

define('AUTH_KEY',         'VALUES COPIED FROM THE COMMAND LINE');
define('SECURE_AUTH_KEY',  'VALUES COPIED FROM THE COMMAND LINE');
define('LOGGED_IN_KEY',    'VALUES COPIED FROM THE COMMAND LINE');
define('NONCE_KEY',        'VALUES COPIED FROM THE COMMAND LINE');
define('AUTH_SALT',        'VALUES COPIED FROM THE COMMAND LINE');
define('SECURE_AUTH_SALT', 'VALUES COPIED FROM THE COMMAND LINE');
define('LOGGED_IN_SALT',   'VALUES COPIED FROM THE COMMAND LINE');
define('NONCE_SALT',       'VALUES COPIED FROM THE COMMAND LINE');

. . .
```

Next, we are going to modify some of the database connection settings at the beginning of the file. You need to adjust the database name, the database user, and the associated password that you configured within MySQL.

The other change we need to make is to set the method that WordPress should use to write to the filesystem. Since we've given the web server permission to write where it needs to, we can explicitly set the filesystem method to "direct". Failure to set this with our current settings would result in WordPress prompting for FTP credentials when we perform some actions.

This setting can be added below the database connection settings, or anywhere else in the file:

/var/www/wordpress/wp-config.php

```
. . .

// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/** MySQL database username */
define( 'DB_USER', 'wordpressuser' );

/** MySQL database password */
define( 'DB_PASSWORD', 'password' );

/** MySQL hostname */
define( 'DB_HOST', 'localhost' );

/** Database Charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8' );

/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );


. . .

define('FS_METHOD', 'direct');
```

Save and close the file when you are finished.

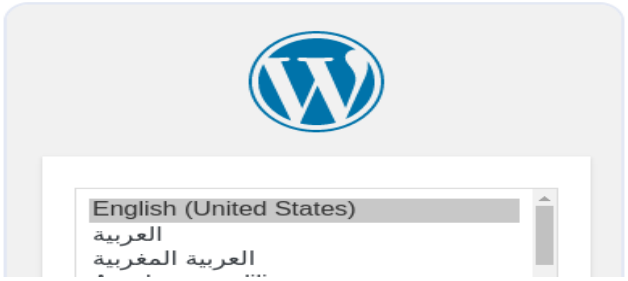
Step 6 – Completing the Installation Through the Web Interface

Now that the server configuration is complete, we can complete the installation through the web interface.

In your web browser, navigate to your server's domain name or public IP address:

```
https://server_domain_or_IP
```

Select the language you would like to use:



Initial Server Setup with Ubuntu 20.04

Step 1 – Logging in as root

To log into your server, you will need to know your **server's public IP address**. You will also need the password or — if you installed an SSH key for authentication — the private key for the **root** user's account. If you have not already logged into your server, you may want to follow our guide on how to Connect to Droplets with SSH, which covers this process in detail.

If you are not already connected to your server, log in now as the **root** user using the following command (substitute the highlighted portion of the command with your server's public IP address):

```
$ ssh root@your_server_ip
```

Copy

Accept the warning about host authenticity if it appears. If you are using password authentication, provide your **root** password to log in. If you are using an SSH key that is passphrase protected, you may be prompted to enter the passphrase the first time you use the key each session. If this is your first time logging into the server with a password, you may also be prompted to change the **root** password.

About root

The **root** user is the administrative user in a Linux environment that has very broad privileges. Because of the heightened privileges of the **root** account, you are *discouraged* from using it on a regular basis. This is because the **root** account is able to make very destructive changes, even by accident.

The next step is setting up a new user account with reduced privileges for day-to-day use. Later, we'll show you how to temporarily gain increased privileges for the times when you need them.

Step 2 – Creating a New User

Once you are logged in as **root**, you'll be able to add the new user account. In the future, we'll log in with this new account instead of **root**.

This example creates a new user called **sammy**, but you should replace that with a username that you like:

```
# adduser sammy
```

Copy

You will be asked a few questions, starting with the account password.

Enter a strong password and, optionally, fill in any of the additional information if you would like. This is not required and you can just hit `ENTER` in any field you wish to skip.

Step 3 – Granting Administrative Privileges

Now we have a new user account with regular account privileges. However, we may sometimes need to do administrative tasks.

To avoid having to log out of our normal user and log back in as the **root** account, we can set up what is known as *superuser* or **root** privileges for our normal account. This will allow our normal user to run commands with administrative privileges by putting the word `sudo` before the command.

To add these privileges to our new user, we need to add the user to the **sudo** group. By default, on Ubuntu 20.04, users who are members of the **sudo** group are allowed to use the `sudo` command.

As **root**, run this command to add your new user to the **sudo** group (substitute the highlighted username with your new user):

```
# usermod -aG sudo sammy
```

Copy

Now, when logged in as your regular user, you can type `sudo` before commands to run them with superuser privileges.

Step 4 – Setting Up a Basic Firewall

Ubuntu 20.04 servers can use the UFW firewall to make sure only connections to certain services are allowed. We can set up a basic firewall using this application.

Note: If your servers are running on DigitalOcean, you can optionally use DigitalOcean Cloud Firewalls instead of the UFW firewall. We recommend using only one firewall at a time to avoid conflicting rules that may be difficult to debug.

Applications can register their profiles with UFW upon installation. These profiles allow UFW to manage these applications by name. OpenSSH, the service allowing us to connect to our server now, has a profile registered with UFW.

You can see this by typing:

```
# ufw app list
```

Copy

```
Output
Available applications:
  OpenSSH
```

We need to make sure that the firewall allows SSH connections so that we can log back in next time. We can allow these connections by typing:

```
# ufw allow OpenSSH
```

Copy

Afterwards, we can enable the firewall by typing:

```
# ufw enable
```

Copy

Type `y` and press `ENTER` to proceed. You can see that SSH connections are still allowed by typing:

```
# ufw status
```

Copy

```
Output
Status: active

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

As **the firewall is currently blocking all connections except for SSH**, if you install and configure additional services, you will need to adjust the firewall settings to allow traffic in. You can learn some common UFW operations in our *UFW Essentials* guide.

Step 5 – Enabling External Access for Your Regular User

Now that we have a regular user for daily use, we need to make sure we can SSH into the account directly.

Note: Until verifying that you can log in and use `sudo` with your new user, we recommend staying logged in as **root**. This way, if you have problems, you can troubleshoot and make any necessary changes as **root**. If you are using a DigitalOcean Droplet and experience problems with your **root** SSH connection, you can regain access to Droplets using the Recovery Console.

The process for configuring SSH access for your new user depends on whether your server's **root** account uses a password or SSH keys for authentication.

If the root Account Uses Password Authentication

If you logged in to your **root** account *using a password*, then password authentication is *enabled* for SSH. You can SSH to your new user account by opening up a new terminal session and using SSH with your new username:

```
$ ssh sammy@your_server_ip
```

Copy

After entering your regular user's password, you will be logged in. Remember, if you need to run a command with administrative privileges, type `sudo` before it like this:

```
$ sudo command_to_run
```

Copy

You will be prompted for your regular user password when using `sudo` for the first time each session (and periodically afterwards).

To enhance your server's security, **we strongly recommend setting up SSH keys instead of using password authentication**. Follow our guide on setting up SSH keys on Ubuntu 20.04 to learn how to configure key-based authentication.

If the root Account Uses SSH Key Authentication

If you logged in to your **root** account *using SSH keys*, then password authentication is *disabled* for SSH. You will need to add a copy of your local public key to the new user's `~/.ssh/authorized_keys` file to log in successfully.

Since your public key is already in the **root** account's `~/.ssh/authorized_keys` file on the server, we can copy that file and directory structure to our new user account in our existing session.

The simplest way to copy the files with the correct ownership and permissions is with the `rsync` command. This will copy the **root** user's `.ssh` directory, preserve the permissions, and modify the file owners, all in a single command. Make sure to change the highlighted portions of the command below to match your regular user's name:

> **Note:** The `rsync` command treats sources and destinations that end with a trailing slash differently than those without a trailing slash. When using `rsync` below, be sure that the source directory (`~/.ssh`) **does not** include a trailing slash (check to make sure you are not using `~/.ssh/`).
>
> If you accidentally add a trailing slash to the command, `rsync` will copy the *contents* of the **root** account's `~/.ssh` directory to the `sudo` user's home directory instead of copying the entire `~/.ssh` directory structure. The files will be in the wrong location and SSH will not be able to find and use them.

```
# rsync --archive --chown=sammy:sammy ~/.ssh /home/sammy
```

Copy

Now, open up a new terminal session on your local machine, and use SSH with your new username:

```
$ ssh sammy@your_server_ip
```

Copy

You should be logged in to the new user account without using a password. Remember, if you need to run a command with administrative privileges, type `sudo` before it like this:

```
$ sudo command_to_run
```

Copy

Install wordpress on windaw iis server

Step 1: Download WordPress

1. Go to the official WordPress website (https://wordpress.org/download/) and download the latest version of WordPress in ZIP format.

Step 2: Extract the WordPress Files

1. Extract the contents of the ZIP file you downloaded to a directory on your Windows server where you want to host your WordPress site. This will be your website's root directory (e.g., `C:\inetpub\wwwroot\mywordpress`).

Step 3: Create a MySQL Database

1. You'll need a MySQL database to store WordPress data. Install MySQL on your Windows server if you haven't already.
2. Create a new MySQL database for WordPress. You can use a MySQL client like phpMyAdmin or the MySQL command line to create the database.
3. Create a MySQL user and grant it privileges on the WordPress database. Note down the database name, username, and password for later use.

Step 4: Configure PHP on IIS

1. Install PHP on your Windows server. You can use a package like XAMPP or install PHP manually.
2. Configure PHP to work with IIS by adding PHP as a FastCGI module. You can use the "Internet Information Services (IIS) Manager" to configure this.

Step 5: Configure WordPress

1. Rename the `wp-config-sample.php` file in your WordPress directory to `wp-config.php`.
2. Open `wp-config.php` and configure the database settings by entering the database name, username, and password you created earlier:

```php
define('DB_NAME', 'your_database_name');
define('DB_USER', 'your_database_user');
define('DB_PASSWORD', 'your_database_password');
```

3. You may also want to define other constants like the authentication keys and salts. You can generate these keys by visiting the WordPress.org secret key generator (https://api.wordpress.org/secret-key/1.1/salt/).
4. Save your `wp-config.php` file.

Step 6: Configure IIS for WordPress

1. Open the "Internet Information Services (IIS) Manager."
2. In the Connections pane on the left, expand your server node and click on "Sites."
3. Right-click on "Default Web Site" or create a new site if needed, and select "Add Application."
4. Set an alias for your WordPress site (e.g., "mywordpress") and browse for the physical path where you extracted the WordPress files.
5. In the "Edit Application" dialog, go to the "Modules" section, and ensure that the "FastCgiModule" is installed and enabled.

Step 7: Access Your WordPress Site

1. Open a web browser and enter the URL to your WordPress site (e.g., `http://localhost/mywordpress`). Follow the WordPress installation instructions.
2. Complete the installation by providing your site title, username, password, and email address.
3. Once the installation is complete, you can log in to the WordPress admin dashboard and start customizing your site.

That's it! You've successfully installed WordPress on your Windows IIS server. You can now start building and managing your website using the WordPress platform.