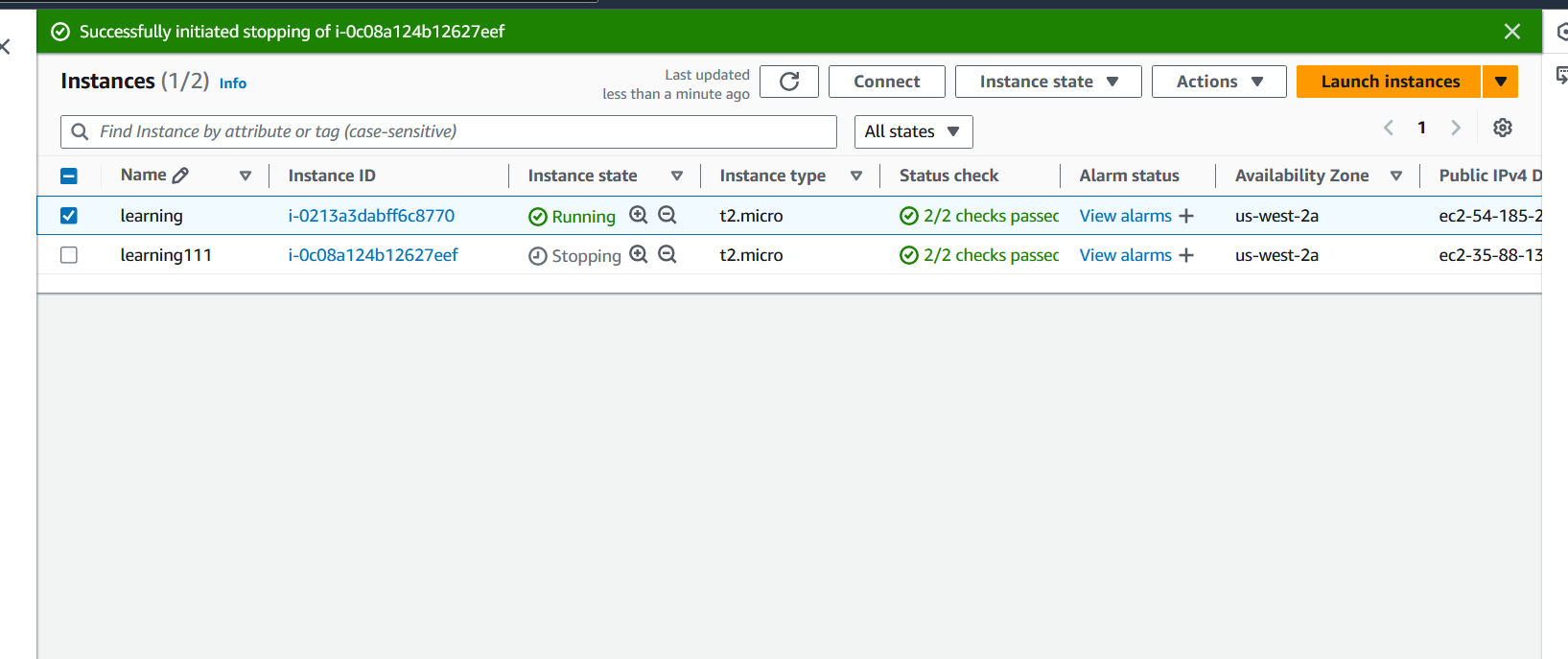
# Setting Up a LAMP Stack on Ubuntu EC2 Instance

**Link to Github Documentation:**

https://github.com/Macbeth111/DevOps-Learning.git

# Step 1: Launching an EC2 Instance

The first step was to launch an EC2 instance from the t3.micro family with Ubuntu Server (HMV). I named the instance "learning" and selected the AMI: ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20240801.



# Step 2: SSH Access to the Instance

After launching the EC2 instance, I learned how to SSH into it using Windows PowerShell CLI. To do this, I used the private key (.pem) that was generated during the instance setup, and connected by running the SSH command with the public IP address of my EC2 instance.

*A screenshot of a computer screen

Description automatically generated*

In addition, I learned to SSH into my Ubuntu server using Putty. I retrieved my server’s IP address by running the command "ip a". However, I encountered issues with SSH being inactive on my server.

To resolve the issue, I first ran a series of commands to update the package list, install the OpenSSH server, start and enable SSH, and check its status. I run the following codes in the order that they are listed:

*sudo apt update*

*sudo apt install openssh-server*

*sudo systemctl start ssh*

*sudo systemctl enable ssh*

*sudo systemctl status ssh*

Despite these steps, SSH was still inactive. I ran another command to check the installation and re-installed OpenSSH, after which I restarted the SSH service.

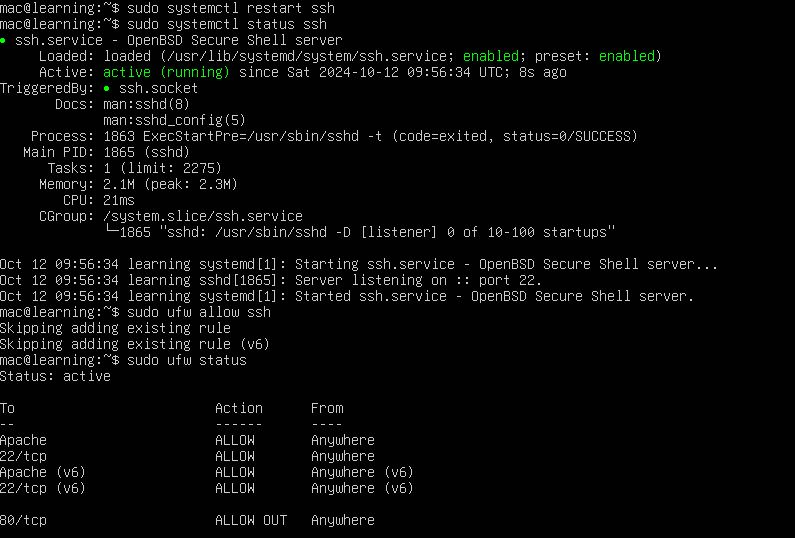
*sudo apt list --installed | grep openssh-server*

*sudo apt install openssh-server*

Then I had to restart SSH using the following code:

*sudo systemctl restart ssh*

Then I checked SSH Stats again using 'sudo systemctl status ssh' and realized it had become active.



After this, I finally accessed Putty successfully using SSH

A computer screen shot of a black screen

Description automatically generated

# Step 3: Installing Apache and updating Firewall

I used Ubuntu package manager 'apt' to install Apache

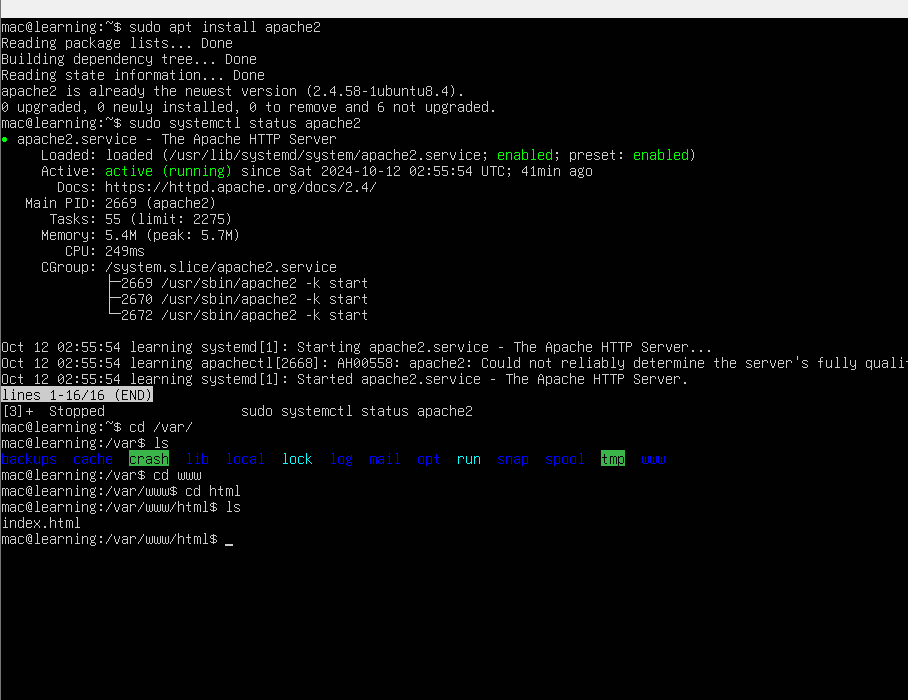
I run the following codes:

*$ sudo apt update*

*$sudo apt install apache2*

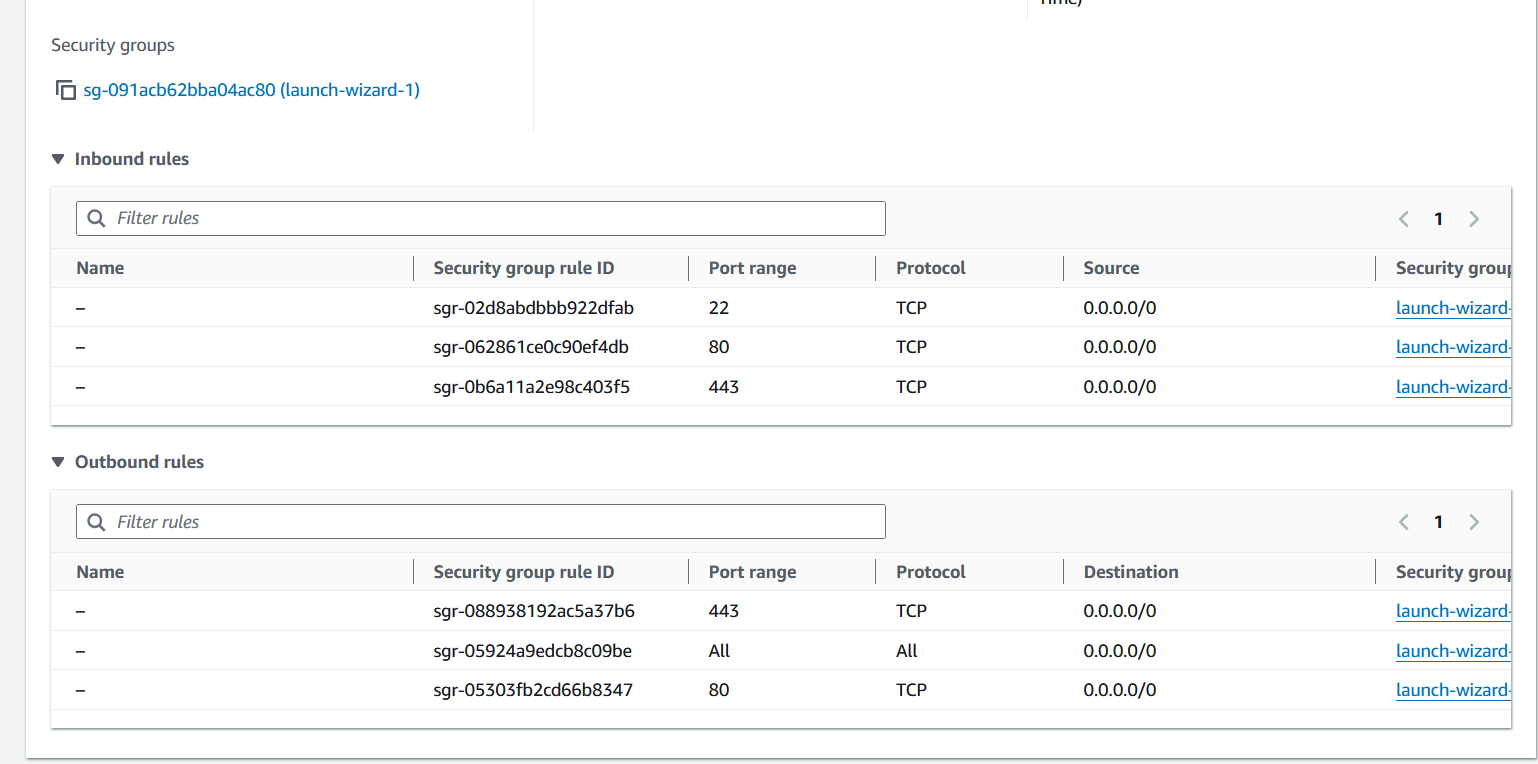
*$sudo systemctl status apache2.*

This ensured that Apache HTTP server was active (running)



After that I went to my EC2 instance and edited inbound rules to receive request from Port 80. Next, I accessed the Apache server via DNS by using the curl command and then updated the inbound rules on my EC2 instance to allow HTTP requests through Port 80.

Then I used the $ curl 'http://localhost:80' command to access the Apache Server via my IP-Address.



A screenshot of a computer

Description automatically generated

# step 4: Installing Mysql-Server

I proceeded to install mysql-server using the code:

*sudo ap install mysql-server*

then i confirmed if it had been properly installed using

*sudo mysql*

*A screenshot of a computer program

Description automatically generated*

Then I set root password for MySQL using

*ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'PassWord.1';*

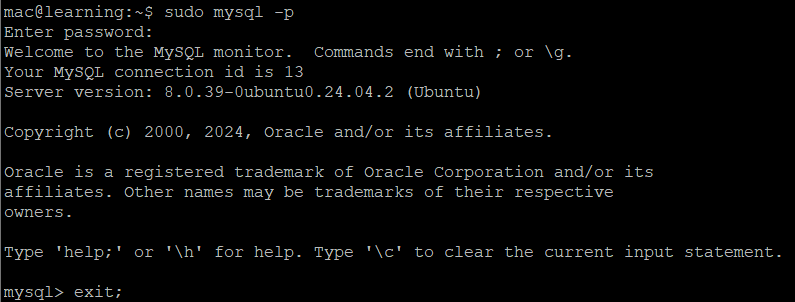
Next, I ran an interactive script that allowed me to set a root password and improve security settings. Afterward, I tested MySQL by logging in with the newly set password and confirmed that everything was working properly.

*sudo mysql\_secure\_installation*

After setting up the password for MySQL I tested it using

*sudo MySQL -p*

I was able to log in successfully.



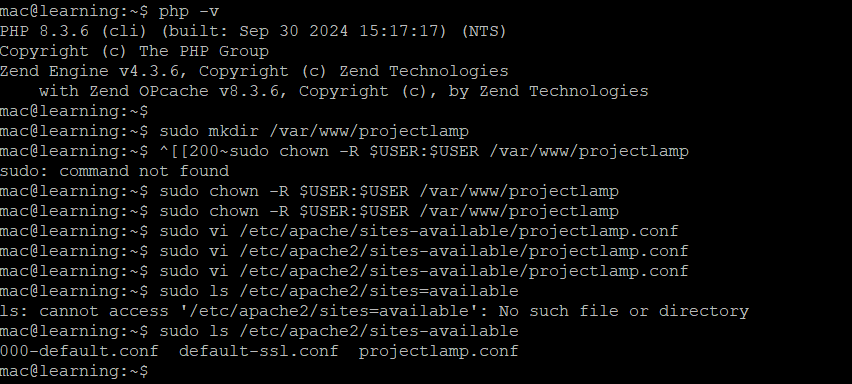
# Step 5: Installing PHP

I understand that PHP is needed to process dynamic web content, while php-mysql allows PHP to communicate with MySQL-based databases, and libapache2-mod-php enables Apache to handle PHP files.

So, I run the command below to install PHP along with the necessary Apache and MySQL modules:

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

After installation, I run : *php -v* to confirm the version.



*AT THIS POINT I UNDERSTAND THAT LAMP IS COMPLETELY INSTALLED AND FULLY OPERATIONAL ON MY UBUNTU SERVER.*

# Step 6: Creating a Virtual Host for the Website

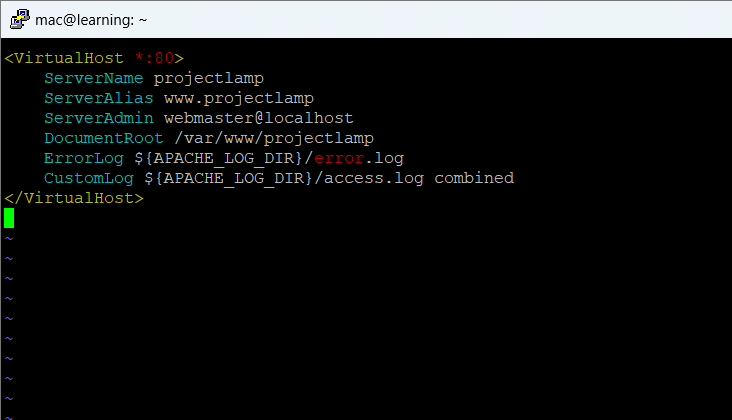
I first created a directory for my project "project lamp" using the command:

*mkdir /var/www/projectlamp*

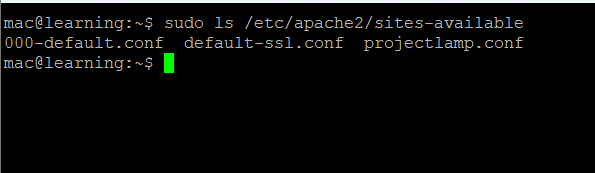
Then I assigned ownership to the directory using the $USER environment variable. The following command was used to do that:

*$ sudo chown -R $USER:$USER /var/www/projectlamp*

Then I opened a new configuration file in Apache's sites-available directory (sudo vi /etc/apache2/sites-available/projectlamp.conf) using vi or vim.



Then I used the list command to show new files in the sites-available directory



I proceeded to enable new virtualhost by using

*sudo a2ensite projectlamp*

and also disabled default website that comes with Apache by using

*sudo a2dissite 000-default*

I checked to see if configuration did not contain errors by using

*sudo apache2ctl configtest*

I then created an html file by pastig in this command:

(NB: I modified the command a bit by inserting my name as the hostname.)

*sudo echo 'Hello LAMP from Macbeth' $(TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"` && curl -H "X-aws-ec2-metadata-token: $TOKEN" -s http://169.254.169.254/latest/meta-data/public-hostname) 'with public IP' $(TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"` && curl -H "X-aws-ec2-metadata-token: $TOKEN" -s http://169.254.169.254/latest/meta-data/public-ipv4) > /var/www/projectlamp/index.html*

Afterward, I listed the files in the sites-available directory to confirm the new virtual host configuration was in place. Next, I enabled the virtual host using the appropriate command and disabled the default website that comes with Apache. I also checked the configuration for any errors and ensured that everything was set up correctly. Finally, I created a custom HTML file to display a test message on my webpage. This message included my hostname and the public IP address of my server. After reloading my webpage, I verified that the setup was successful and the message was displayed correctly.

# step 7: Enabling PHP on the website

The first step in the process was to modify the Apache configuration file to prioritize the loading of PHP files when a directory is accessed. Apache, by default, looks for specific file types (e.g., index.html, index.php) to load when a directory is requested in a browser. To modify this behavior, I needed to edit the directory index settings. I began by opening the dir.conf file, which is located in /etc/apache2/mods-enabled/. This file controls how Apache handles directory indexing. Using the vim editor, I ran the following command in the terminal:

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

This command opens the configuration file for editing with root privileges, which are necessary to make changes to system-level files. Once inside the file, I located the <IfModule mod\_dir.c> section, which lists the files Apache looks for by default. Initially, the configuration listed multiple file types in a specific order, with index.html as the first file. My goal was to prioritize index.php as the first file in this list. The original configuration looked like this:

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

I edited the configuration to make index.php the first file Apache would look for:

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

This change ensures that whenever a directory is accessed, Apache will first look for a file named index.php before checking for other file types like index.html or index.cgi. After making this change, I saved the file and closed the vim editor by pressing Esc and typing *:wq.*

Once the configuration was edited, the next step was to apply the changes by reloading the Apache service. Instead of fully restarting Apache, which could temporarily interrupt the web server, I reloaded the service using the following command:

*sudo systemctl reload apache2*

Reloading the service ensures that the new configuration takes effect without affecting the server’s availability. The next part of the task involved creating a PHP test script to verify that Apache could properly handle and process PHP files. To do this, I created a new file named index.php in the custom web root folder where the website’s files are stored. I executed the following command to open a blank file using vim:

*$ vim /var/www/projectlamp/index.php*

This path points to the directory where the website’s files are located. In this case, /var/www/projectlamp/ is the root folder for the web project. Once the file was opened, I added a simple PHP script to confirm the server’s ability to handle PHP. The script I used calls the phpinfo() function, which provides detailed information about the PHP environment on the server. The code I added to index.php was:

*<?php*

*phpinfo();*

*?>*

The phpinfo() function is commonly used to test PHP installations because it displays essential information, such as the PHP version, installed modules, and configuration settings.After adding the PHP code, I saved and closed the file by pressing Esc and typing :wq in the vim editor. The next step was to test whether Apache was correctly processing PHP files. To do this, I opened a web browser and navigated to the server’s IP address or domain, followed by /index.php.

*http://http://192.168.185.98/index.php*

The page displayed PHP details, including the version, enabled modules, and server configuration. This successful display of the PHP info page indicated that both Apache and PHP were properly configured. The changes I made to prioritize index.php ensured that the server would load PHP files first when a directory is accessed, which is critical for PHP-based web applications.

A screenshot of a computer

Description automatically generated

# Challenges and Solutions

Throughout this process, I faced some challenges, particularly with SSH access and ensuring the proper configuration of the LAMP stack. One significant issue was that SSH was initially inactive, requiring me to reinstall and restart the SSH service to resolve the problem. Another challenge involved configuring Apache to prioritize PHP files, which I overcame by editing the correct configuration file.

To overcome these challenges, I reached out to colleagues for assistance, and my colleague Fisayo was particularly helpful. I am grateful for this opportunity.