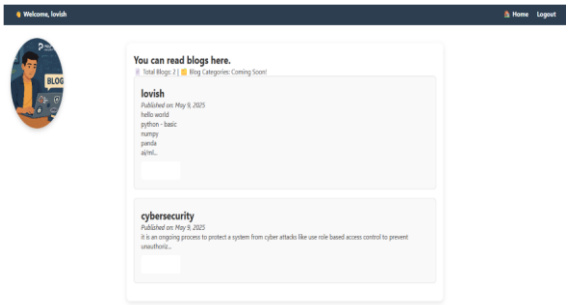
Assignment-internetworking security

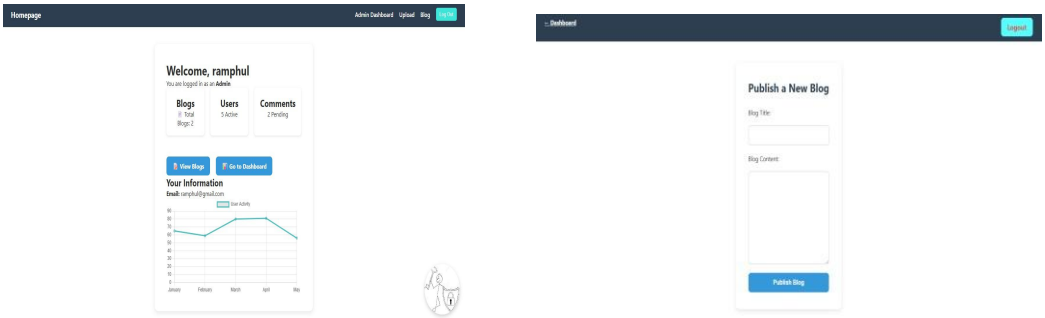
Group members: Deansingh Ramphul Kushalsing Bhantooa

Contents

Introduction about the website:

This project involves setting up a personal portfolio website on a virtual machine running Ubuntu. The website includes a login system and blog functionality, using a MySQL database to store user and blog data. The main goal is to deploy the site securely, then simulate and test common cybersecurity attacks such as SQL injection, brute-force login attempts, and unauthorized access to admin pages. This helps in understanding how vulnerabilities can be exploited and how to defend against them using secure coding practices and tools like firewalls (iptables).



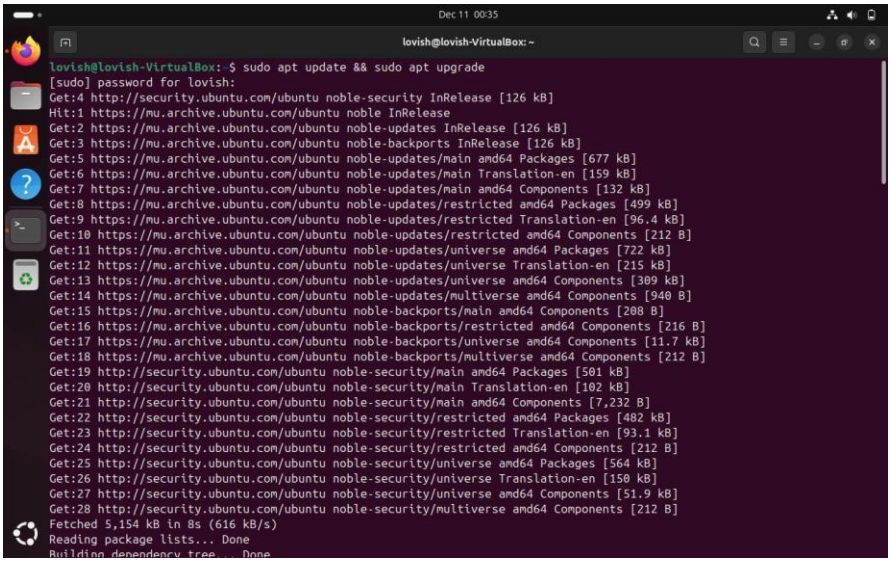


This is our website.

Part 1: Website and Web Server Setup

Step 1: Update and Upgrade System Packages

Before installing any software, I made sure the system's package list and installed packages were up to date.



Explanation:

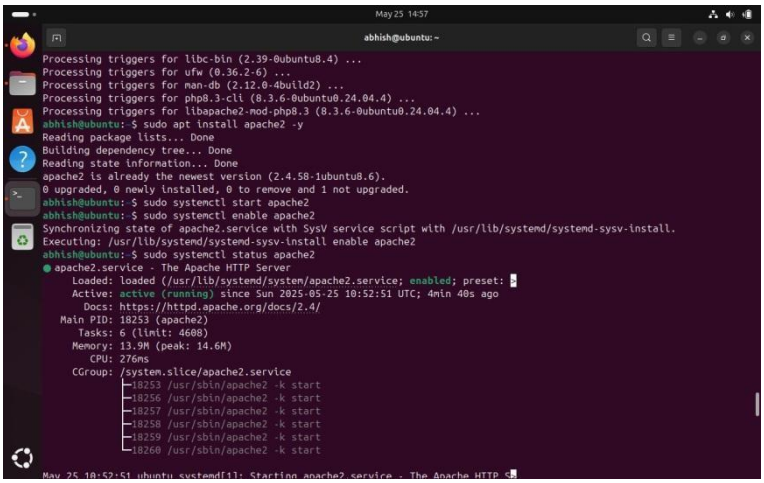
- sudo apt update:
  This command updates the list of available packages and versions from the Ubuntu repositories. It does **not install** or upgrade any packages — it just refreshes the list.

- sudo apt upgrade:
  This command installs the **newest versions** of all packages that are currently installed on the system. It upgrades the software to make sure everything is up to date and secure.

 Figure 1: Running sudo apt update && sudo apt upgrade to update the system.

Step 2: Install Apache Web Server

Apache is the software that serves the website over the internet.

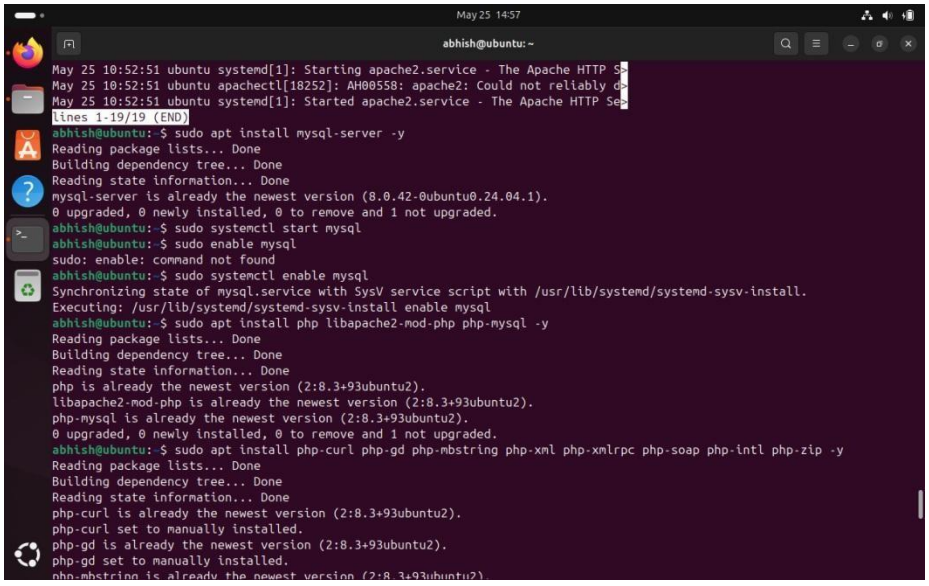


- apache2: The Apache web server package.
- -y: Automatically confirms the installation.

 Figure 2: Apache installed successfully.

Step 3: Install MySQL Server

MySQL is the database system used to store website data like users and blogs.

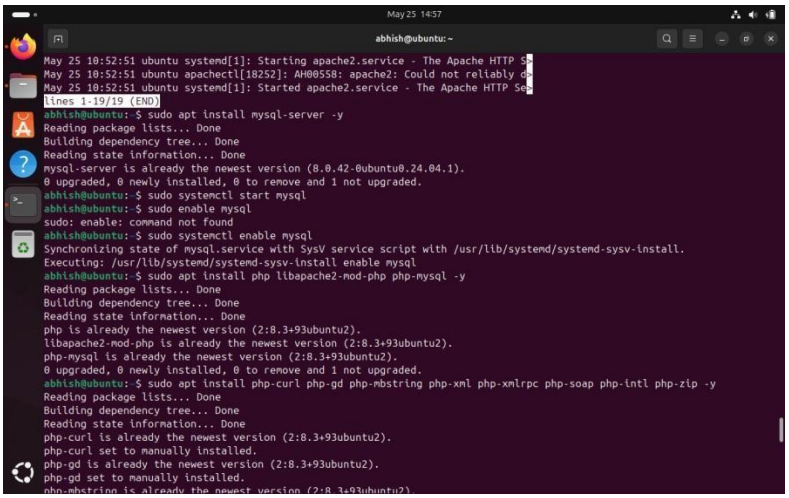


- Installs the MySQL database server.

 Figure 4: MySQL server installation completed.

Step 4: Install PHP

PHP is a programming language used to create dynamic web pages.



- php: Installs the core PHP package.
- libapache2-mod-php: Allows Apache to run PHP files.

Step 5: Install PHP-MySQL Extension

This package allows PHP to connect and communicate with MySQL databases.

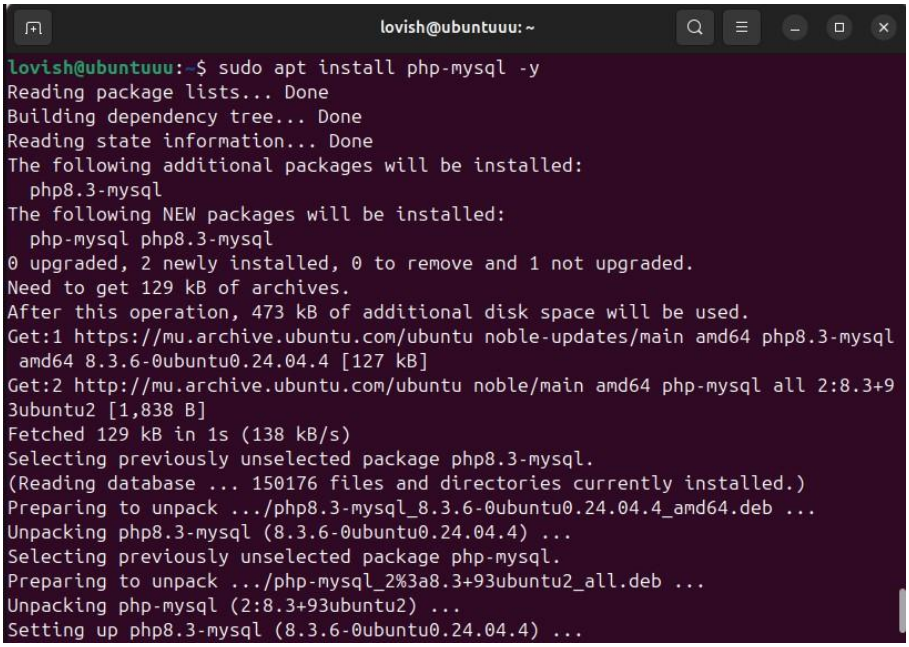


 Figure 5: PHP-MySQL module installed to connect PHP with MySQL **Step**

6: Install phpMyAdmin phpMyAdmin is a web-based interface to manage

MySQL databases easily.

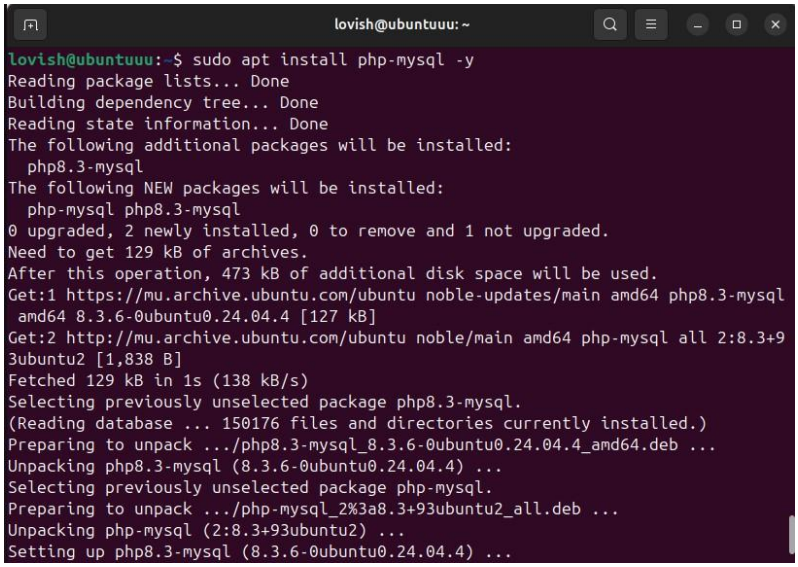


 Figure 6: phpMyAdmin installed.

Step 7: Secure the MySQL Root User

I logged into MySQL and changed the root password for security and phpMyAdmin compatibility.

 Figure 7: MySQL root user secured with native password.

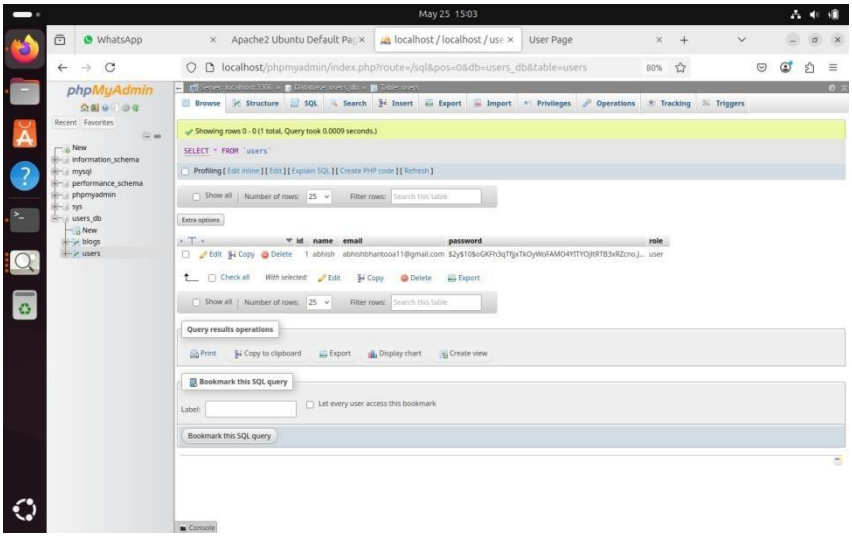
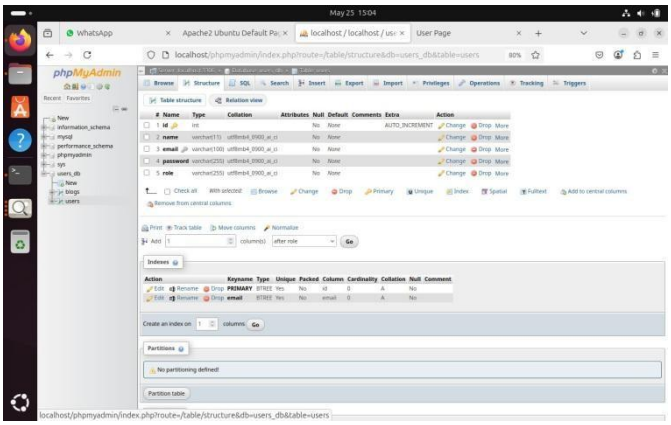
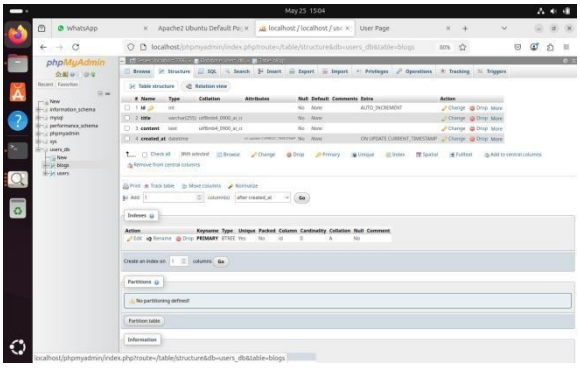


Figure 8: Database



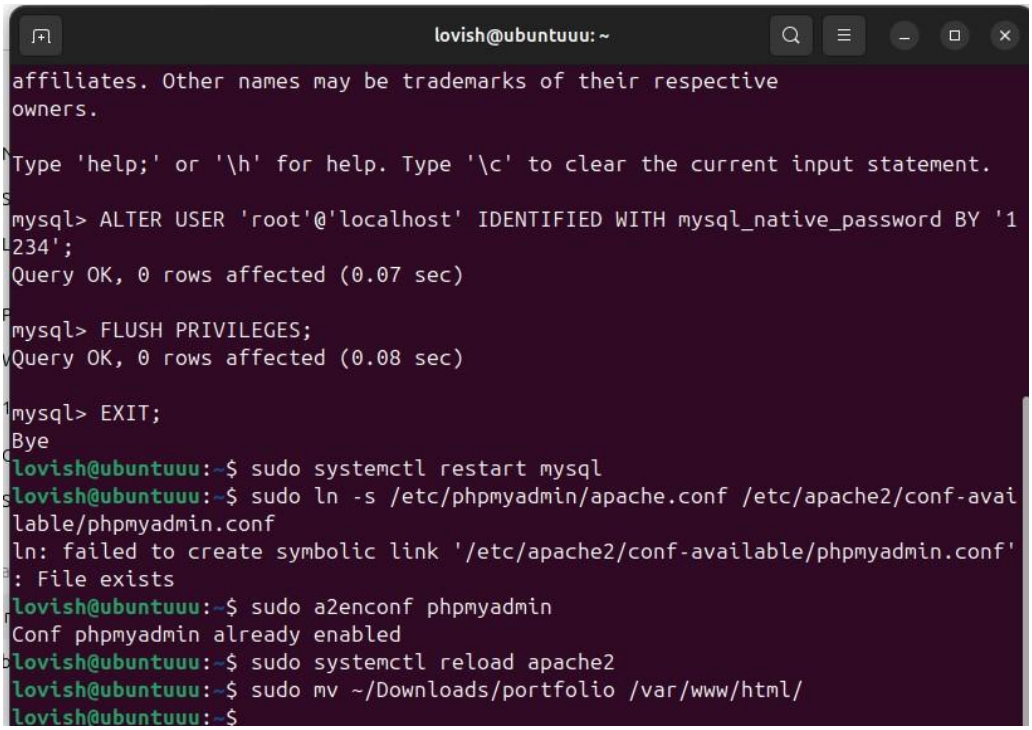
Users table



Blogs table

Step 8: Move Website to Apache Directory and Set Permissions

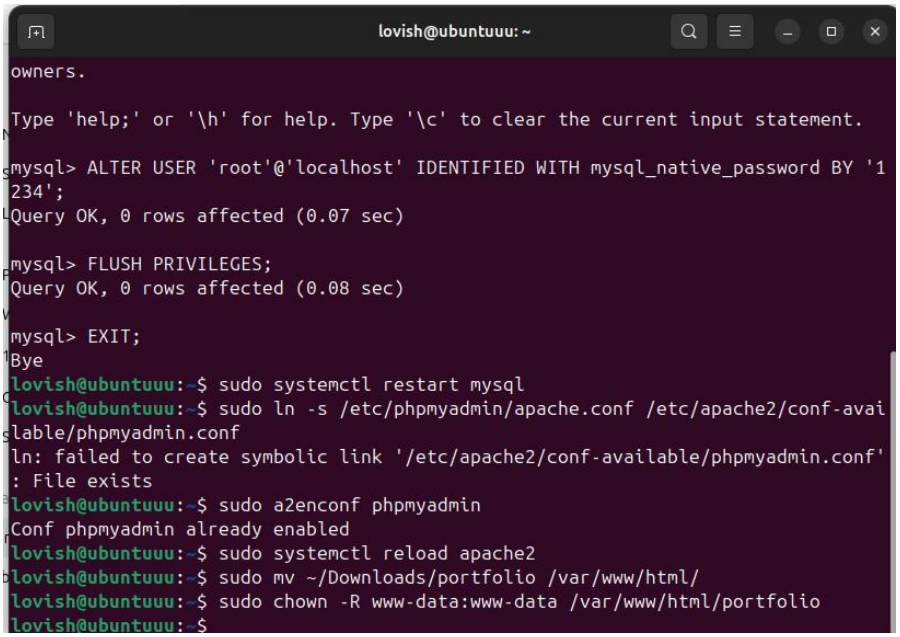
After preparing the website files, I moved the portfolio folder to Apache's web root, set the correct ownership and permissions, and then reloaded Apache to apply the changes.



- **sudo mv**: Moves the portfolio directory from the Downloads folder to /var/www/html/, which is Apache's default web directory.

Step 9: Set Ownership of Website Files

To ensure that Apache has the correct permissions to read and serve the website files, I changed the ownership of the entire portfolio directory to the Apache user and group.



- **sudo chown**: Changes the owner of files or directories.

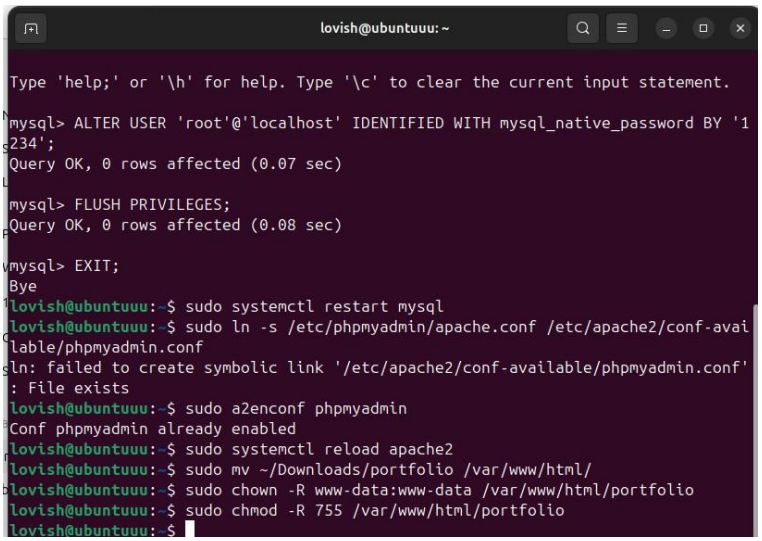
- **-R**: Applies the ownership change **recursively** to all files and subdirectories.

This step is important to prevent **403 Forbidden** errors and to allow Apache to properly access the website files.

Figure 9: Ownership of the portfolio directory set to www-data for secure web server access.

Step 10: Set File Permissions for Website Directory

After setting the ownership, I configured the appropriate permissions on the portfolio directory to ensure that Apache could read and execute the website files securely.



- **sudo chmod**: Changes file and directory permissions.
- **-R**: Applies the permissions **recursively** to all files and subdirectories.
- **755**:
    - Owner (www-data) → Read, write, and execute

○ Group and others → Read and execute only This step is

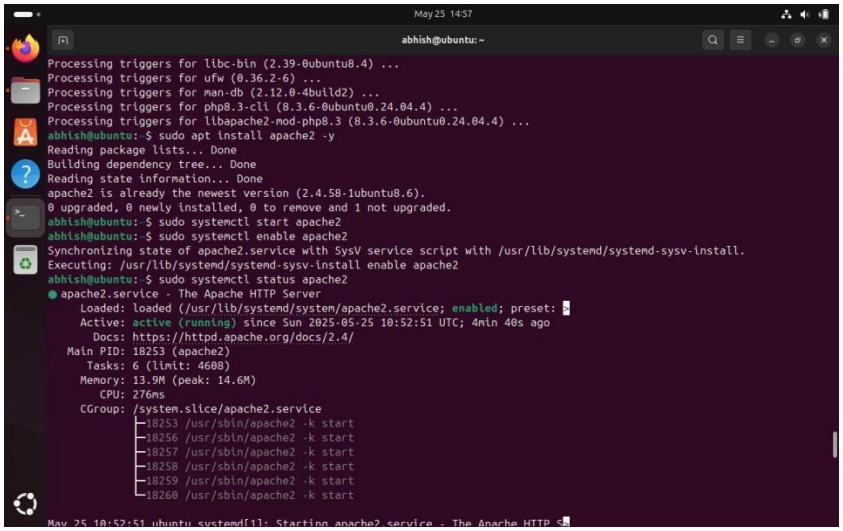
critical for maintaining **security and functionality**:

- Apache can execute and read the files.
- Others can access the website content, but **cannot modify** it.

 Figure 10: File permissions set to 755 for all files and folders inside the portfolio directory.

Step 11: Reload Apache to Apply Changes

After setting the correct file ownership and permissions, I reloaded Apache to apply any changes without interrupting ongoing connections.



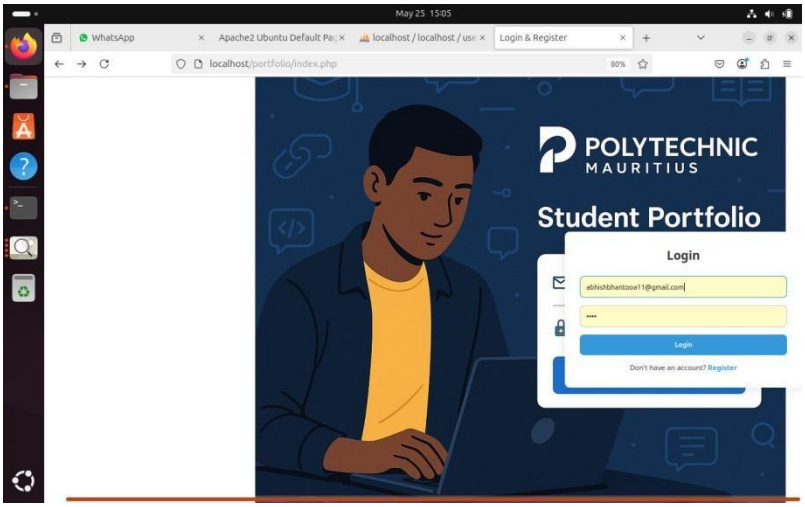
- **sudo systemctl**: Manages system services (like Apache).
- **reload apache2**: Reloads Apache configuration and updates the file structure changes (like new folders or permissions) **without stopping the server**.

Reloading is a safe way to apply changes without downtime.

Figure 11: Apache server reloaded to apply updated ownership and permissions for the portfolio site.

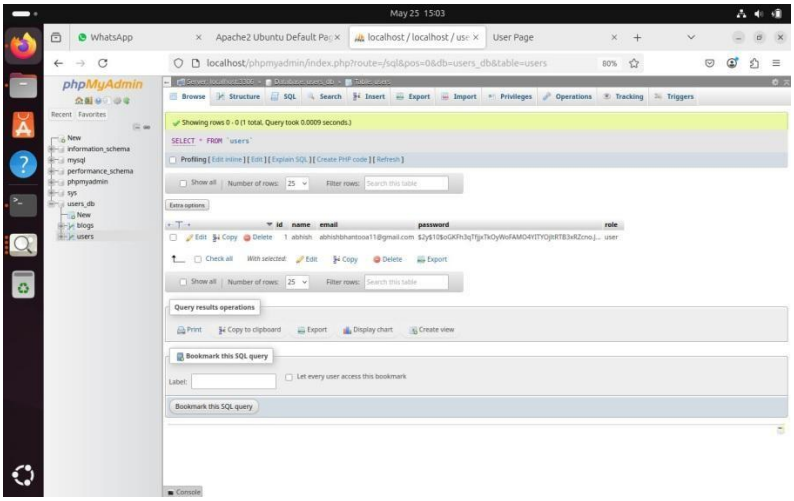
Step 12: Website Access

Portfolio Site:
http://192.168.100.197/portfolio



phpMyAdmin Interface:

http://192.168.100.197/phpmyadmin



Part 2: Implement Access Control Feature

Implement Role-Based Access Control (RBAC)

To ensure that users only access content they're authorized for, I implemented **Role-Based Access Control (RBAC)** using PHP. Each user is assigned a role during registration, and upon login, the user is redirected to a different page depending on their role.

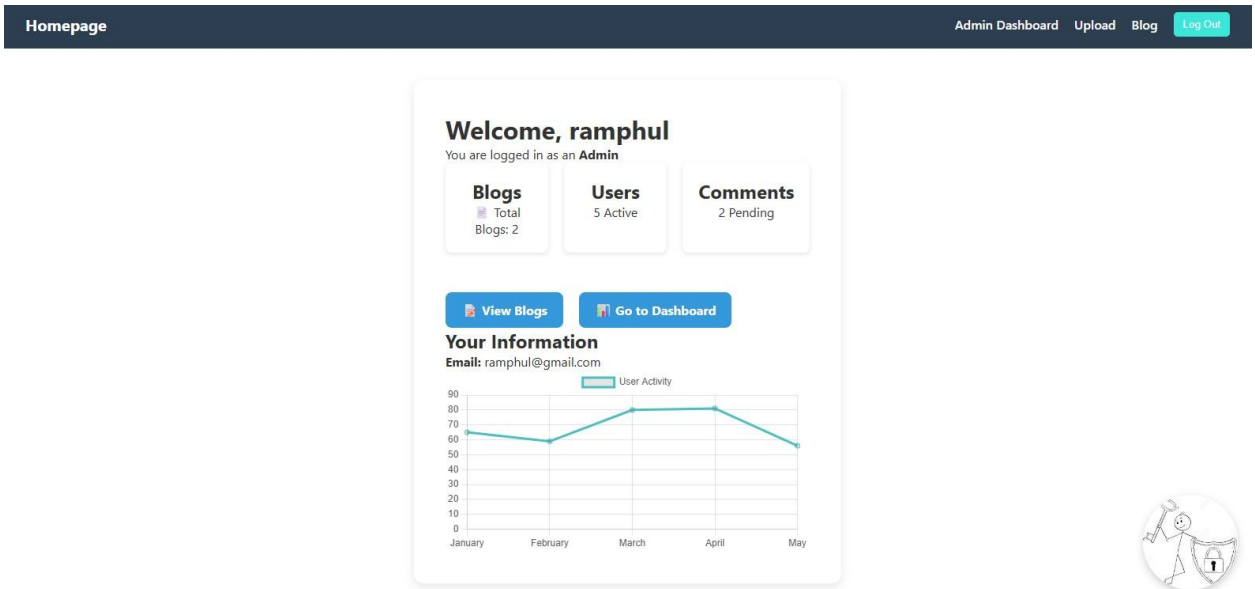
```php
$name = $_POST['name']; // Get name input from the form
$email = $_POST['email']; // Get email input from the form
$password = password_hash($_POST['password'], PASSWORD_DEFAULT); // Hash the password securely
$role = $_POST['role']; // Get selected user role from the form
```

✓ When a user registers, their role (admin or user) is saved in the database.

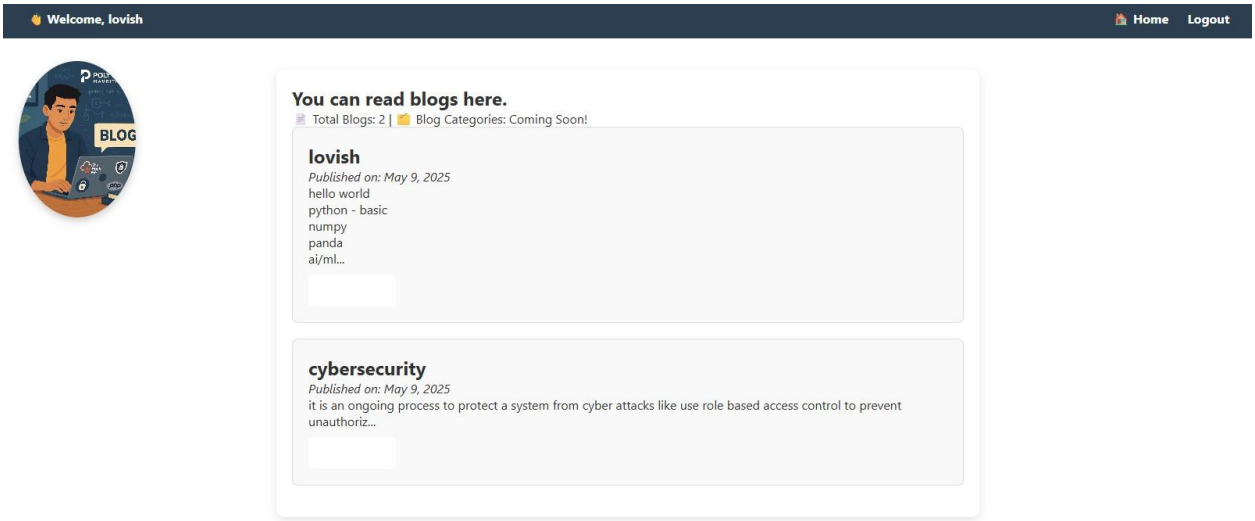
```php
$user = $result->fetch_assoc(); // Fetch user data as associative array
if (password_verify($password, $user['password'])) {
    // If password is correct, store user info in session
    $_SESSION['name'] = $user['name'];
    $_SESSION['email'] = $user['email'];

    // Redirect user based on their role
    if ($user['role'] === 'admin') {
        header("Location: admin_page.php");
    } else {
        header("Location: user_page.php");
    }
    exit(); // Stop further script execution
}
```

After login:

- **Admins** are redirected to admin_page.php



- **Regular users** are redirected to user_page.php

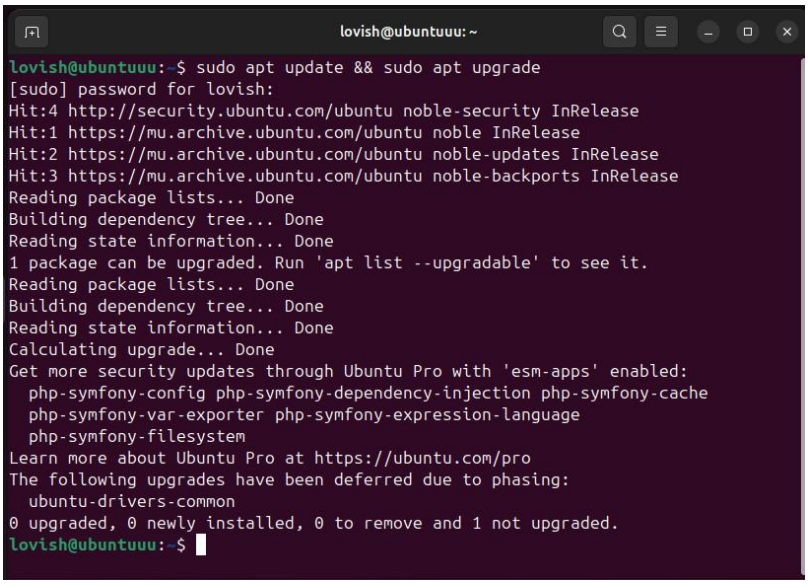


Part 3: Firewall Configuration Using iptables

To secure the server, I configured the firewall using iptables. The goal was to **restrict incoming traffic** and only allow essential services such as HTTP (port 80), HTTPS (port 443), and SSH (port 22) from a specific IP address.

✅ Step 1: Update and Upgrade System Packages

Before making firewall changes, I ensured the system was up to date.



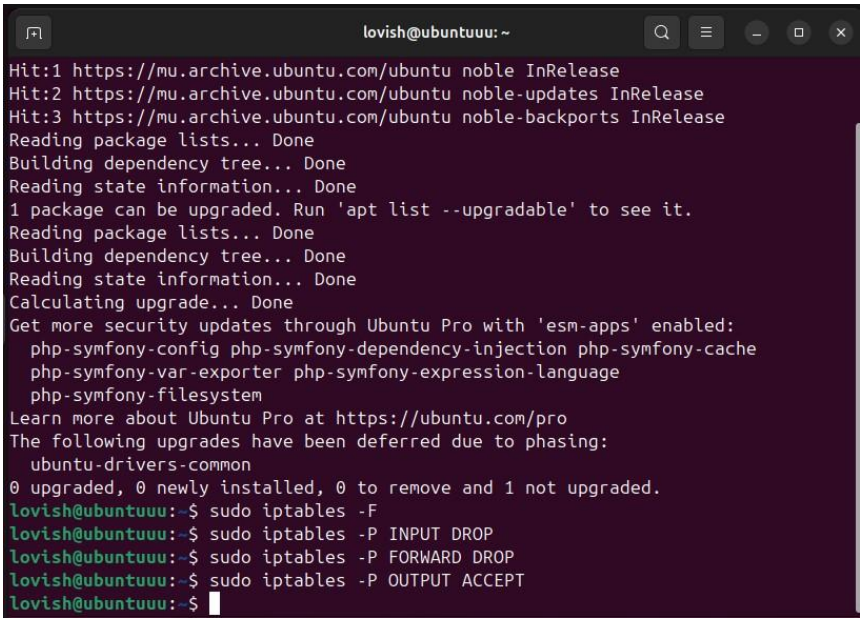
 sudo apt update: Updates the list of available packages. sudo apt upgrade: Installs the latest versions of all installed packages.

Step 2: Flush Existing iptables Rules

Cleared all existing firewall rules to start with a clean configuration.

Step 3: Set Default Policies

Configured the default iptables behavior to **deny incoming and forwarded traffic** but **allow outgoing** connections.



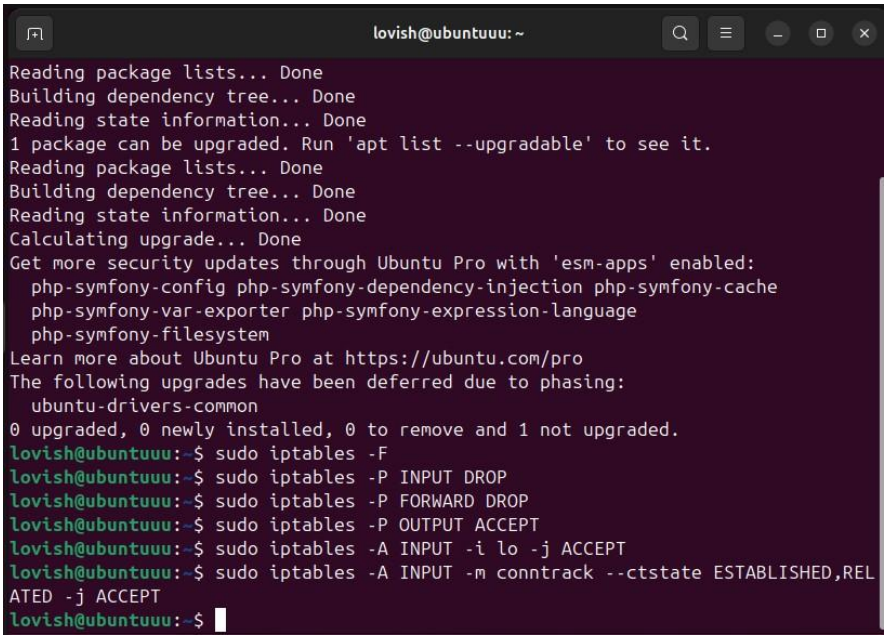
- INPUT DROP: Deny all incoming traffic by default.
- FORWARD DROP: Deny forwarding traffic.
- OUTPUT ACCEPT: Allow all outgoing connections.

Step 4: Allow Localhost (Loopback) Access

Enabled internal communications on the system.

Step 5: Allow Established and Related Connections

Permits return traffic from outgoing connections and connections related to already established sessions.



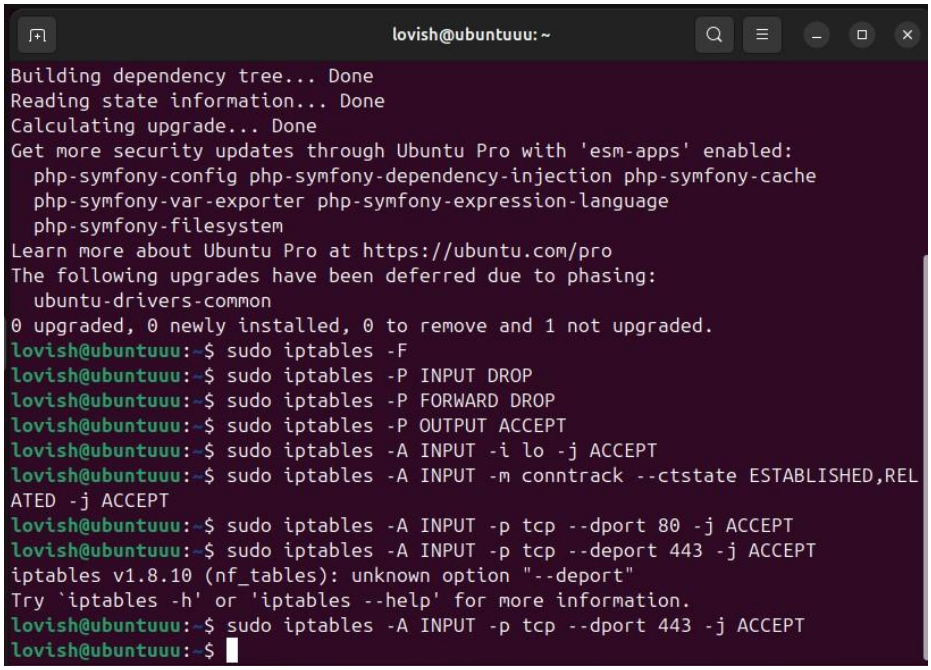
- -i lo: Targets the loopback interface (127.0.0.1).
- -j ACCEPT: Allows the traffic.
- Uses conntrack module to allow traffic that's part of an existing or related connection. Step 6:

  Allow HTTP Traffic (Port 80)

Allows access to the web server via HTTP.

Step 7: Allow HTTPS Traffic (Port 443)

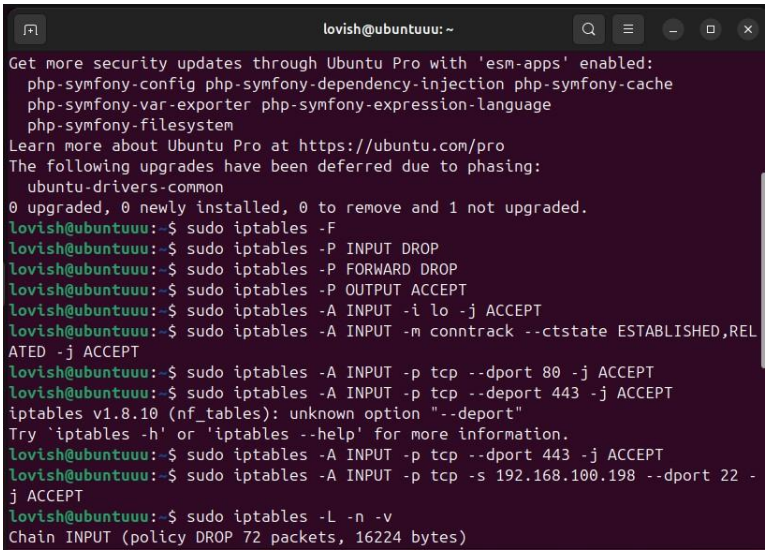
Allows encrypted traffic (HTTPS) to reach the web server.



- --dport 80: Targets TCP traffic on port 80.
- --dport 443: Targets TCP traffic on port 443.

Step 8: Allow SSH Access From Specific IP

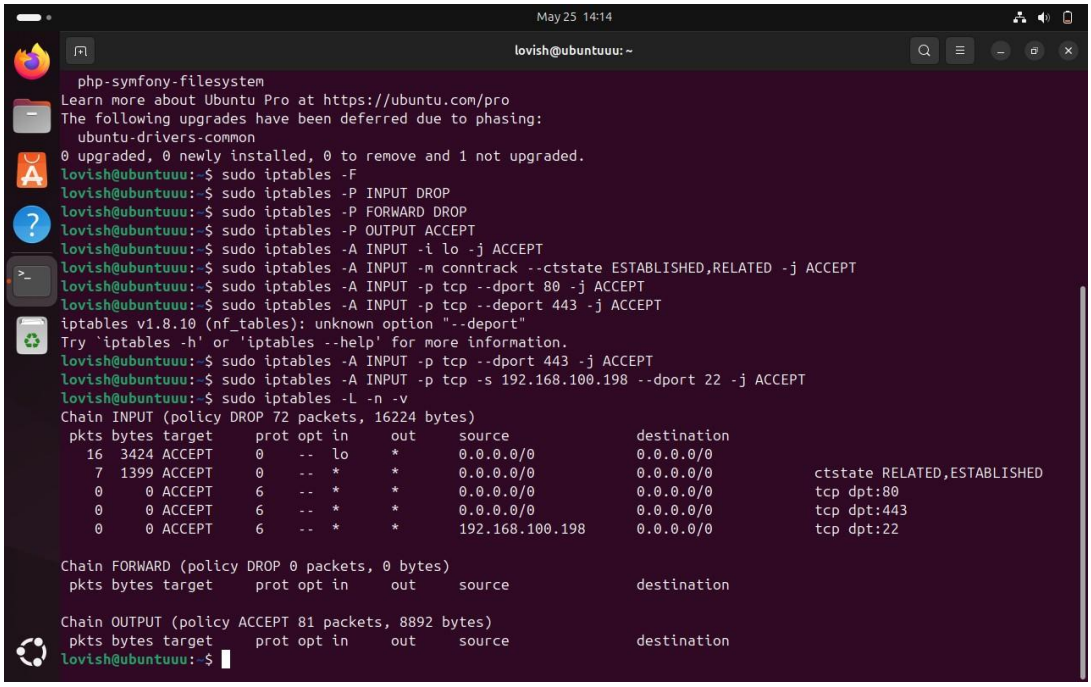
Restricts SSH access to only one trusted IP address.



- -s 192.168.100.198: Only allows SSH from this IP address. --dport 22: SSH port.

Step 9: View Current Firewall Rules

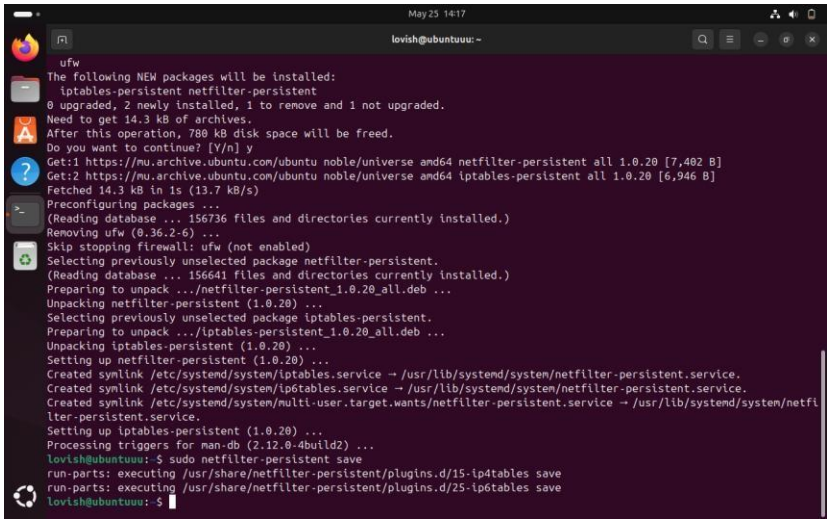
To confirm the firewall rules are set correctly, I listed them.



- -L: Lists all rules.
- -n: Shows numeric values (no DNS). □ -v: Verbose output.

Step 10: Save iptables Rules

Ensures the iptables configuration is preserved after a system reboot.



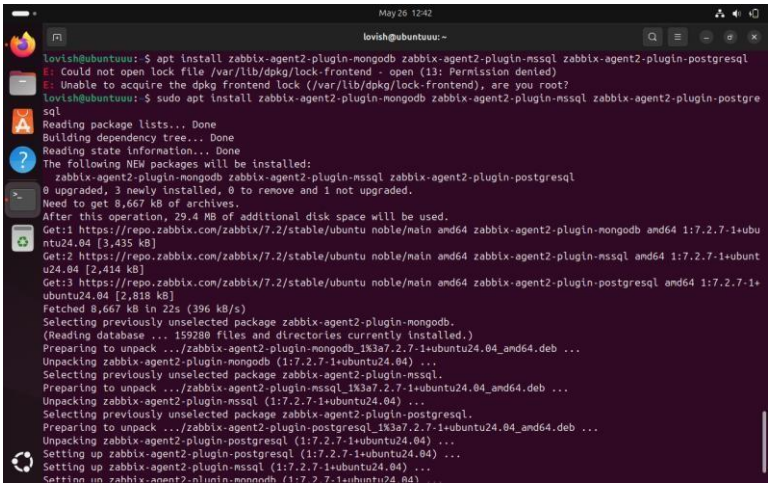
- Saves current iptables rules using the netfilter-persistent service.

Part 4: Server Monitoring with Nagios or RABBIX

Installation of Zabbix Agent

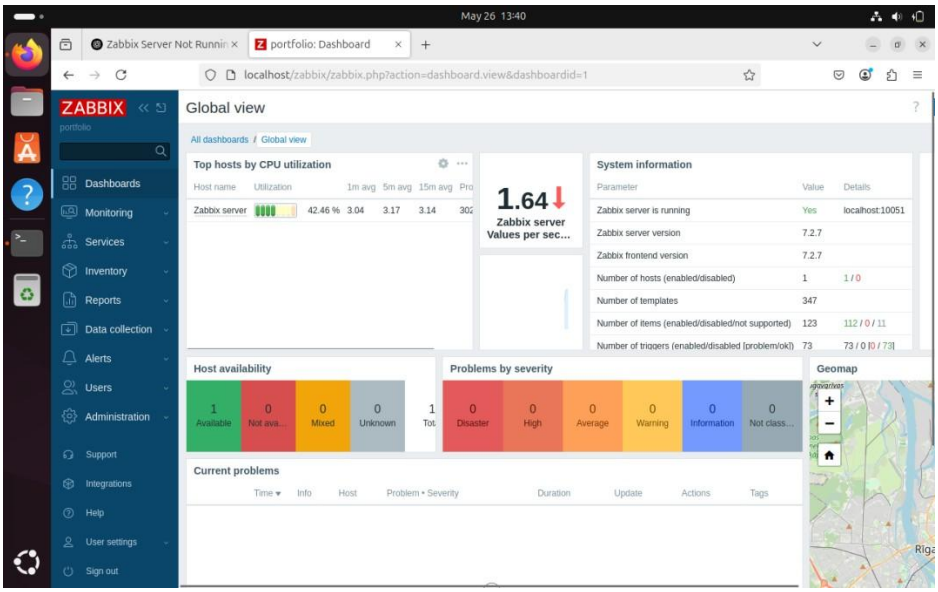
To monitor the website server, I installed the Zabbix agent, which collects system and service data and sends it to the Zabbix monitoring server.

The installation was done using the following commands on the web server:



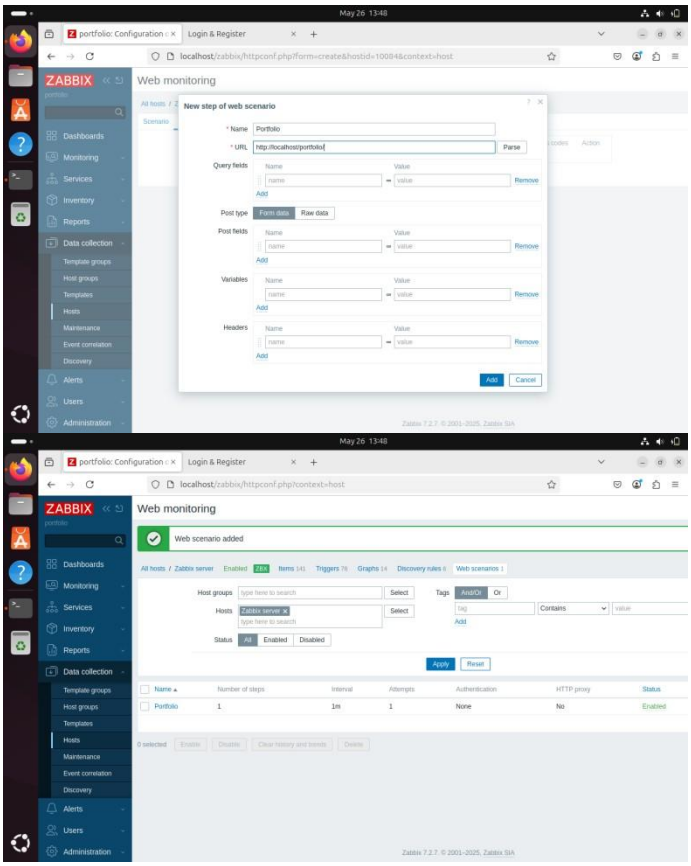
Zabbix Monitoring Dashboard

This screenshot displays the Zabbix dashboard monitoring the web server. It visualizes important performance metrics and the status of services to ensure the website remains available and responsive.



Linking Zabbix Agent to Monitor the Website Server

This screenshot shows the configuration of the Zabbix agent on the web server. The agent is responsible for collecting server performance data and sending it to the Zabbix monitoring server, enabling continuous monitoring of the website's health.

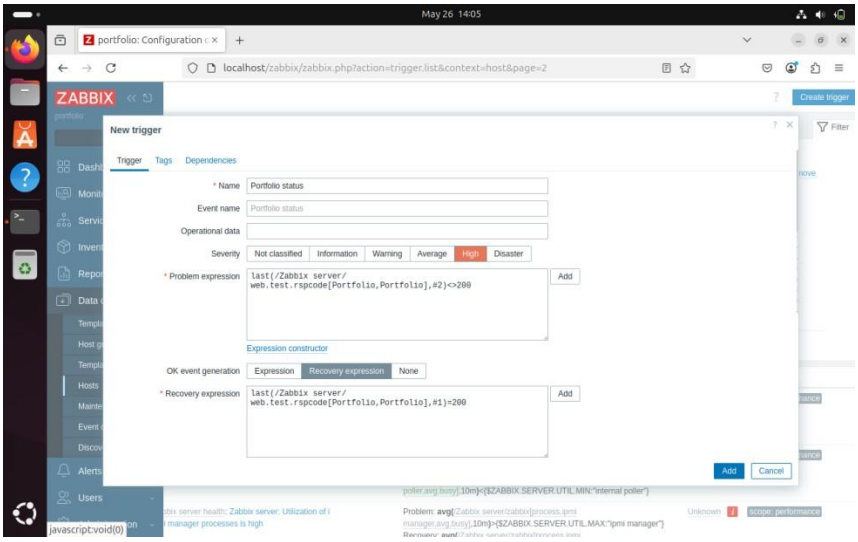


Creating a Trigger to Monitor Website Uptime

This screenshot shows the setup of a trigger in Zabbix to monitor the website's HTTP response status.

- The trigger checks if the website returns a **200 OK** status, indicating the site is up and functioning correctly.
- If the response changes to **404 Not Found** or any other error status, the trigger activates an alert.

This helps to quickly detect when the website is down or unreachable, ensuring prompt action can be taken to restore service.



Trigger for CPU Load and RAM Usage

This screenshot shows the configuration of a combined trigger monitoring both CPU load and RAM usage on the server.

- The trigger monitors the **average system load** and **memory consumption** to detect when the server is under heavy resource usage.
- If the CPU load or RAM usage exceeds the defined thresholds (e.g., CPU load above 80%, RAM usage above 75%), the trigger activates an alert.
- This combined monitoring helps ensure the server remains responsive and prevents performance degradation due to resource exhaustion

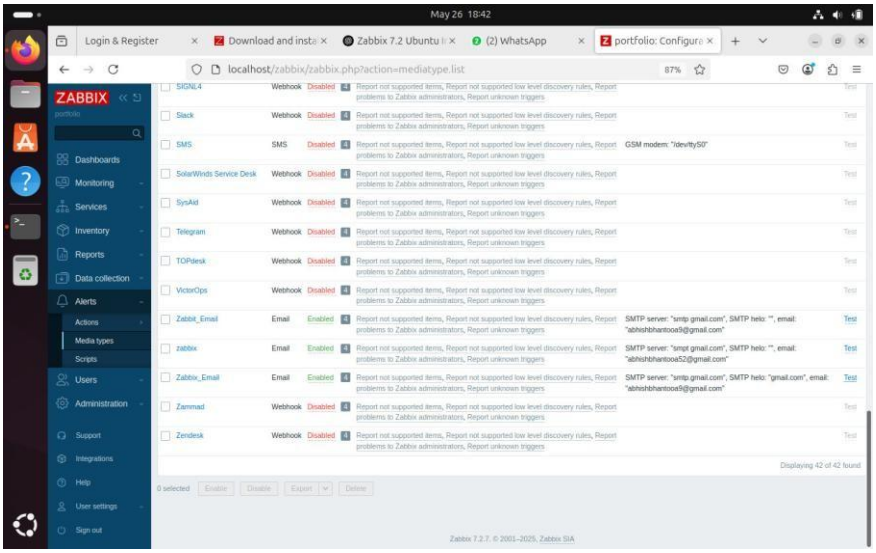
…….

Setting Up Email Alerts for Critical Events

To ensure timely responses to critical server and website issues, email alerts were configured in Zabbix for the following conditions:

- **Website Downtime:** An email notification is sent whenever the website trigger detects downtime, such as receiving a 404 error or no response, enabling immediate investigation and resolution.
- **High CPU/Memory/Disk Usage:** Alerts are triggered and emailed when server resources exceed predefined thresholds (e.g., CPU load over 80%, memory usage above 75%, or disk usage reaching critical levels), helping to prevent performance degradation or system failure.

This alert system allows administrators to proactively monitor server health and maintain website availability by responding quickly to issues as they arise.



Task 5: Website Security Testing – SQL Injection & Forced Browsing

To verify the security of my deployed website, I performed basic **attack simulations** such as **SQL injection** and **forced browsing**. These tests were unsuccessful, confirming that the website had some level of defense mechanisms in place.

Step 1: SQL Injection Protection via Prepared Statements

In the login and registration process, I implemented **prepared statements** to prevent SQL injection.

```php
// Use prepared statement to check if the email is already registered (prevents SQL injection)
$checkEmail = $conn->prepare("SELECT email FROM users WHERE email = ?");
$checkEmail->bind_param('s', $email); // Bind email as string parameter
$checkEmail->execute(); // Execute the query
$checkEmail->store_result(); // Store result to check number of rows returned
```

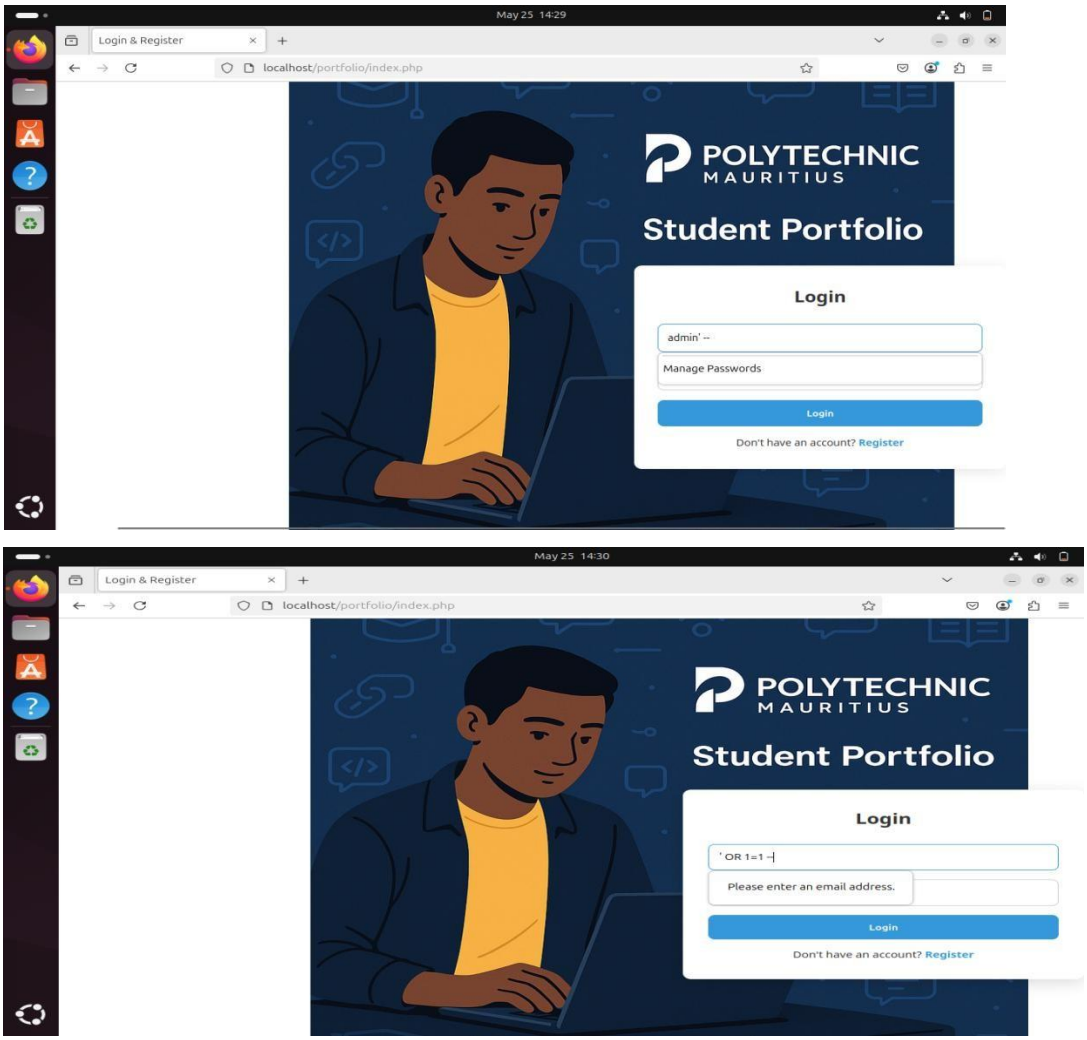
```php
// Use prepared statement to fetch user by email (prevents SQL injection)
$result = $conn->prepare("SELECT * FROM users WHERE email = ?");
$result->bind_param('s', $email); // Bind email as string parameter
$result->execute(); // Execute the query
$result = $result->get_result(); // Get result set from executed statement
```

Explanation:

- **Prepared statements** prevent direct injection of SQL code into the query.
- Inputs are **bound** separately from the SQL logic, protecting against attackers trying to bypass authentication or manipulate the database using strings like admin' OR '1'='1.

Step 2: Simulated SQL Injection Attack (Unsuccessful)

I attempted a SQL injection through the login form using inputs like:



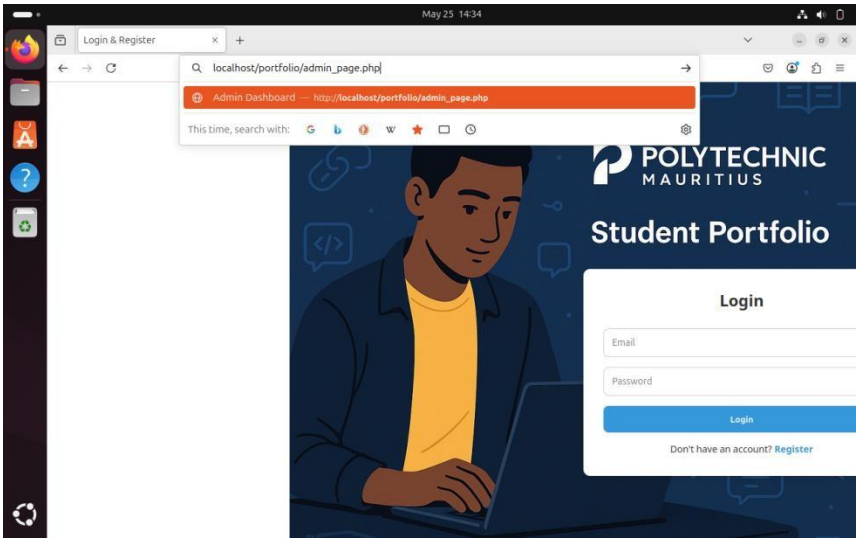
Result: Login failed – due to secure query execution using prepared statements.

Figure: Screenshot showing failed SQL injection login attempt.

Step 3: Forced Browsing Protection (Unsuccessful)

I also tested **forced browsing** by trying to access admin pages directly without logging in.

Example URL tested:



Result: Access denied – the page redirected or blocked access due to lack of session validation.

Figure: Screenshot showing the blocked forced browsing attempt.

Summary of Protection Features

Feature	Status	Protection Type
SQL Injection	Blocked ✓	Prepared Statements (PHP)
Forced Browsing	Blocked ✓	Session-based Access Control

These implementations show that the website has basic but essential security controls against common web attacks.

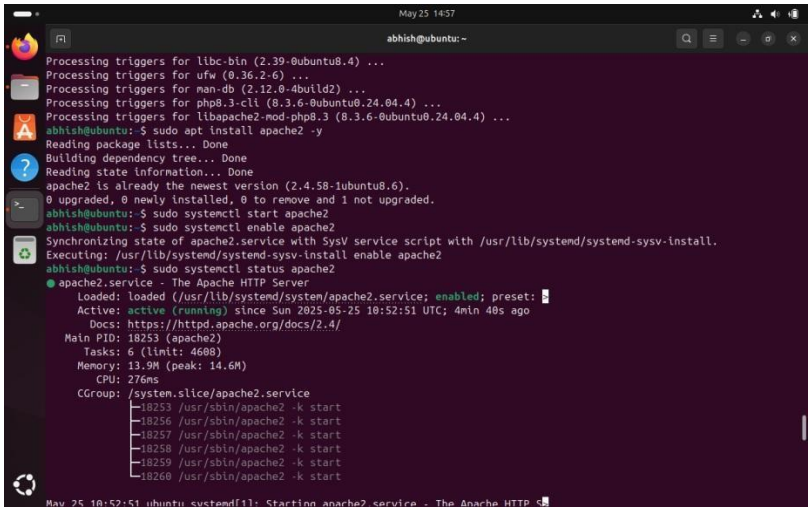
Task 6: Accessing the Website from Another Virtual Machine

After configuring the web server and firewall rules, I tested external access to ensure that the website could be accessed securely from a **different VM within the same network**.

Step 1: Check Apache Service and Firewall Rules Before

testing access from another machine, I made sure:

- Apache service was running:



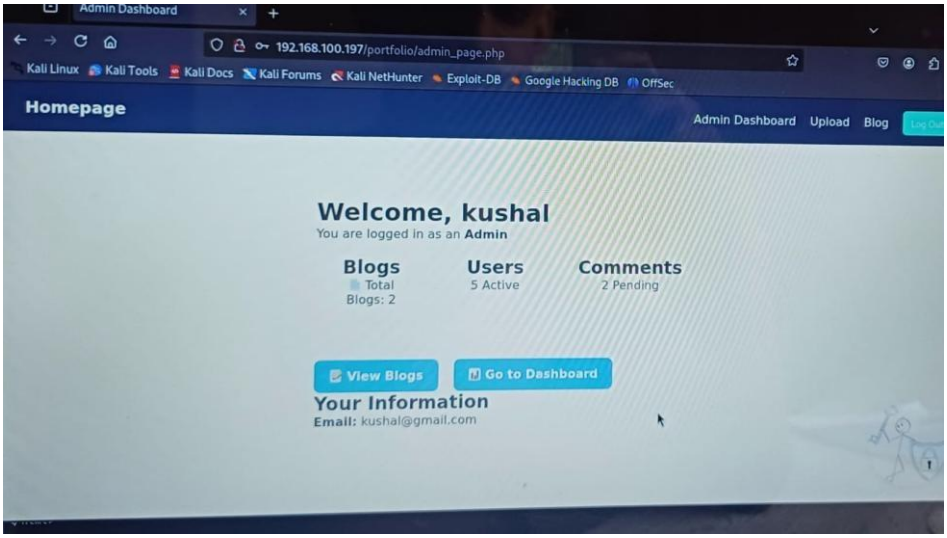
Step 2: Get the Web Server's IP Address

On the hosting VM (Ubuntu server), I retrieved the internal IP address using:

…….

Step 3: Access the Website from Another VM

On a second VM (client), I opened a browser and typed the server IP:



Result: The website loaded successfully, showing that the firewall, Apache configuration, and network settings were correctly applied.

Figure: Screenshot showing the website opened from a second VM.

Summary

Test	Result	Notes
Website accessible via HTTP	✓ Success	Port 80 was open and Apache was running
Firewall configuration	✓ Pass	Only allowed specific traffic (e.g., HTTP, SSH)

Conclusion

Throughout this project, I successfully deployed and secured a web server on Ubuntu by following essential best practices in secure website deployment, access control, firewall configuration, and monitoring.

Key Achievements:

- ✔ Installed and configured Apache to host a portfolio website.
- ✔ Properly set file permissions and ownership to restrict unauthorized access.
- ✔ Implemented **Role-Based Access Control (RBAC)**, allowing users with different roles (admin/user) to access different pages securely.
- ✔ Hardened the server using **iptables**, blocking all unnecessary traffic and allowing only essential ports such as 80 (HTTP) and 22 (SSH).
- ✔ Successfully simulated and **prevented SQL injection and forced browsing** using prepared statements and access checks.
- ✔ Confirmed secure and functional website access from another virtual machine in the network.
- ✔ Monitored server status using **Zabbix**, ensuring ongoing visibility into system health and uptime.

This hands-on deployment reinforced my understanding of secure web hosting, Linux administration, and real-world cybersecurity defenses. It also demonstrated how practical configurations, such as proper user authentication, access restrictions, and firewall rules, can significantly improve the overall security posture of a system.