**Project Name**: Linux Web Server

**Introduction:**

This project involves developing a web application using PHP and MariaDB within a LAMP stack (Linux, Apache, MariaDB, PHP). The primary focus is on building server-side logic, efficient database interactions, and secure, scalable APIs. The project aims to deliver a seamless user experience through dynamic content generation and responsive design, while ensuring data integrity and security.

**Motivation:**

The motivation of this project is to implement the powerful combination of PHP and MariaDB within a LAMP stack to build efficient, scalable, and secure web applications. By focusing on server-side development and database management, the project aims to provide robust solutions that can handle high traffic and complex data requirements. Additionally, the project seeks to enhance user experience through responsive design and dynamic content generation, while ensuring data integrity and security. This approach not only meets current web development standards but also equips developers with essential skills and best practices for future projects.

**Purpose of Work:**

The purpose of this project is to create a web application that efficiently handles data management and server-side processing using PHP and MariaDB within a LAMP stack. The project aims to deliver a seamless user experience through dynamic content generation and responsive design. By focusing on performance optimization, security measures, and scalability, the project seeks to provide a robust solution capable of supporting complex user interactions and high traffic volumes. Additionally, this project aims to facilitate developers in learning and applying best practices in web development and database management.

**Software Required:**

Operating System: Linux

Ubuntu: A popular choice for its user-friendliness and extensive documentation.

Debian: Known for its stability and reliability.

CentOS: Preferred for enterprise environments due to its long-term support.

Fedora: Offers cutting-edge features and the latest software versions.

Web Server: Apache

Apache HTTP Server is one of the most widely used web servers, known for its robustness and flexibility.

Database Server: MariaDB

MariaDB is a popular open-source database server that is a fork of MySQL.

Server-Side Scripting Language: PHP

PHP is a widely-used open-source scripting language especially suited for web development.

**Inputs:**

**User Requests**

* HTTP requests (GET, POST, etc.) from web browsers or API clients.
* Form submissions with user data.

**Database Queries**

* SQL queries to retrieve, insert, update, or delete data in MariaDB.

**Configuration Files**

* Settings for the server, database, and application (e.g., php.ini, my.cnf).

**Codebase**

* PHP scripts and libraries.
* HTML/CSS/JavaScript files for frontend integration.

**External APIs**

* Data from third-party services (e.g., payment gateways, social media APIs).

**Outputs:**

**HTTP Responses**

* HTML pages rendered and sent to the user's browser.
* JSON data for API responses.

**Database Changes**

* Updated, inserted, or deleted records in MariaDB.

**Logs and Errors**

* Logs of application activities and errors for monitoring and debugging.

**User Sessions**

* Managed user sessions for authentication and state maintenance.

**File Storage**

* Uploaded files saved on the server or cloud storage.

**Development and Testing:**

To implement a LAMP (Linux, Apache, MySQL/MariaDB, PHP) server for PHP web development, use the newly released Debian 10 GNU/Linux distribution for the demonstration.

**Updating APT Package Repository Cache:**

First, update the APT package repository cache with the following command:

*$*sudo apt update

The APT package repository cache should be updated.

### **Installing and Configuring MySQL/MariaDB:**

Now, install MariaDB server and client packages from the official package with the following command:

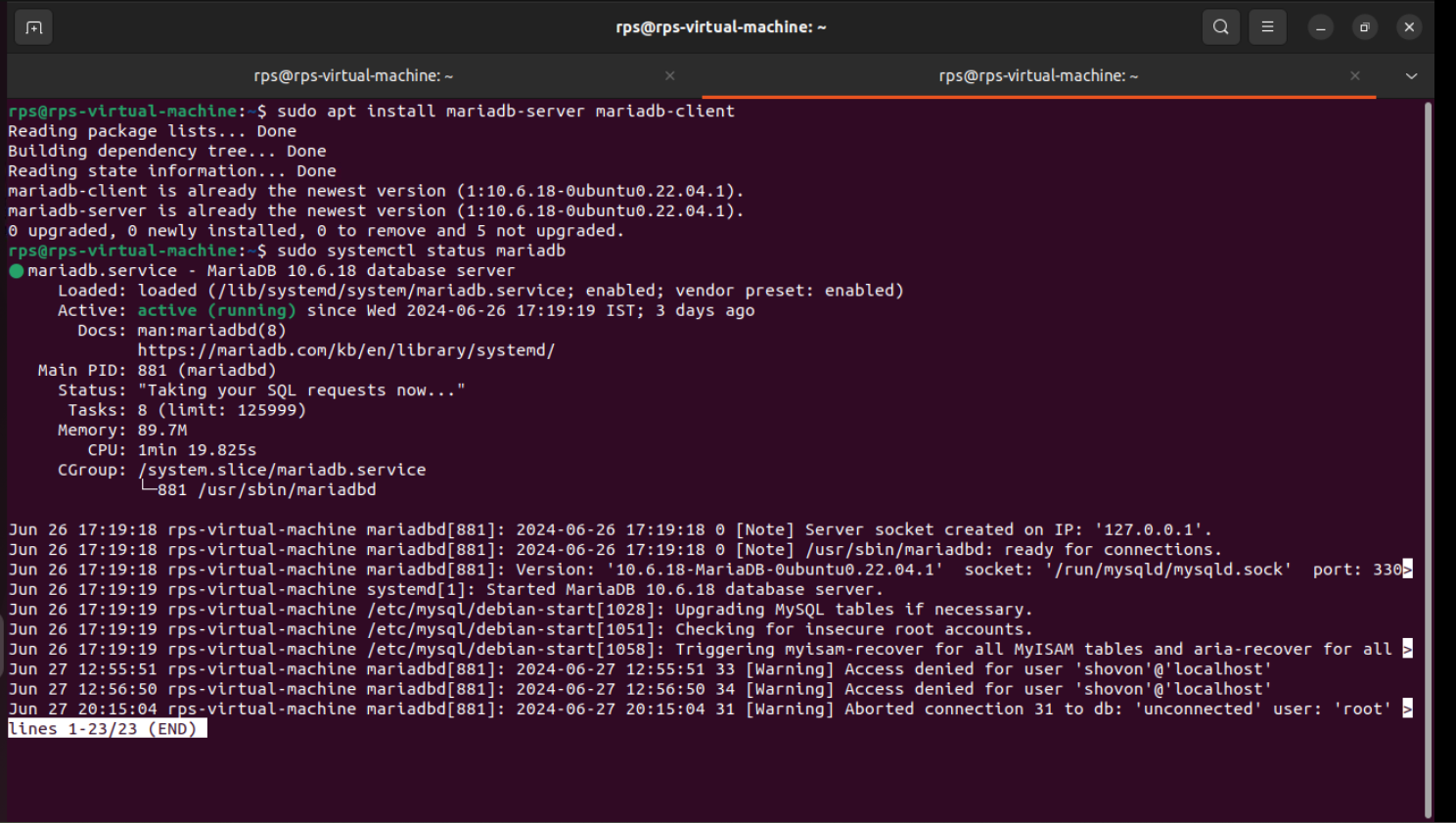
*$*sudo apt install mariadb-server mariadb-client

The APT package manager will download and install all the required packages.

Now, check whether **mariadb** service is running with the following command:

$ sudo systemctl status mariadb

As you can see, the **mariadb** service is running. It’s also **enabled** to automatically start on system boot.

If in any case, **mariadb**service is not running, then start the service with the following command:

*$*sudo systemctl start mariadb

Now, run the following command to set a **root** password for MariaDB:

*$*sudo mysql\_secure\_installation

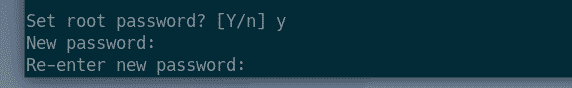
Press **<Enter>**.Now, press **Y** and then press **<Enter>**.

****

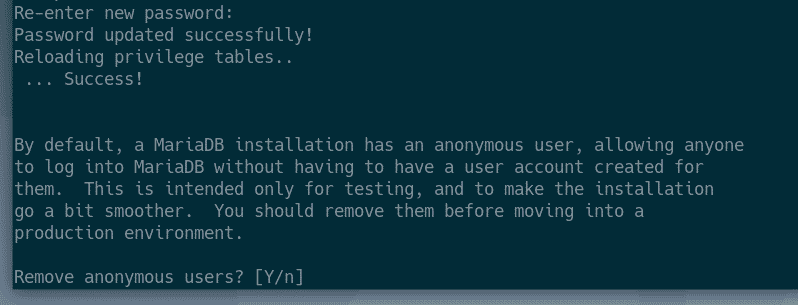
Now, type in your new **root** password and press **<Enter>**.

****

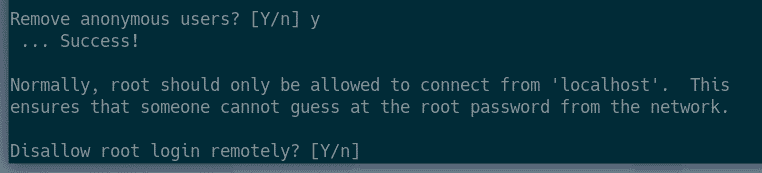
Type in your **root** password again and press **<Enter>**.

****

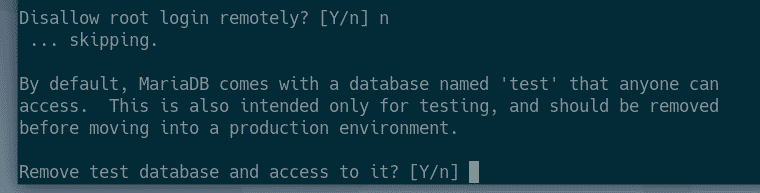
Press **Y** and then press **<Enter>** to remove anonymous users.

****

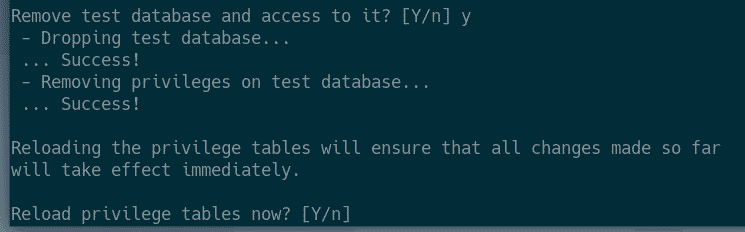
If you don’t want to allow **root** login remotely, press **Y**. Otherwise, press **N**. Then, press **<Enter>**.

****

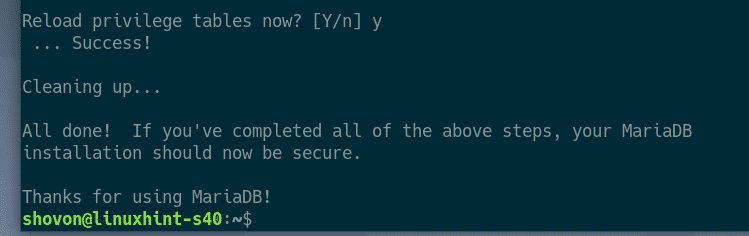
Now, press **Y**and press **<Enter>** to remove test database.

****

Now, press **Y** and then press **<Enter>** to reload the privilege table.

****

MariaDB should be configured.

**Creating New MySQL/MariaDB Users and Databases**:

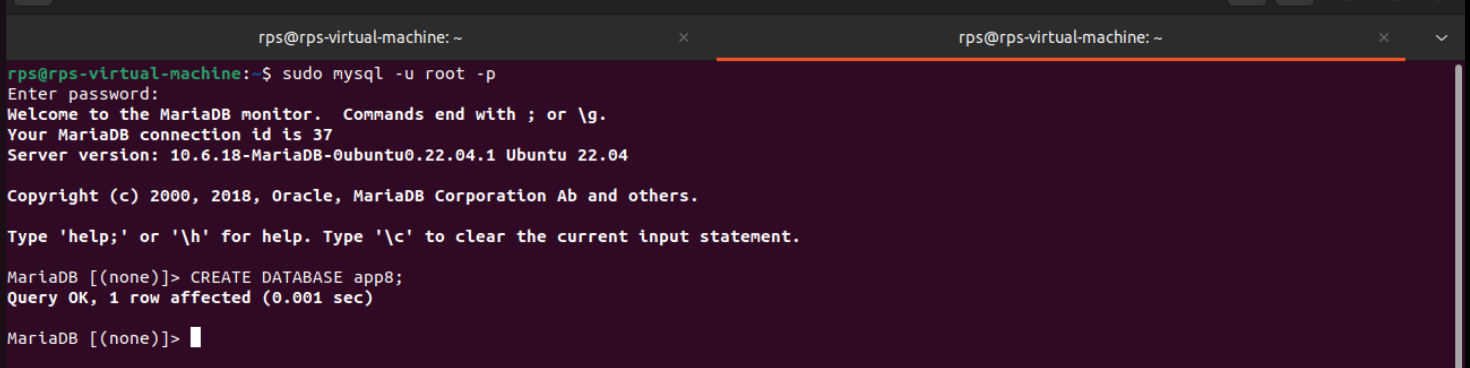
Now, you have to create a new user and a database for your web application.

Login to MariaDB shell with the following command:

*$*sudo mysql -u root -p

Now, type in the MariaDB **root** password you’ve already set and press **<Enter>**.

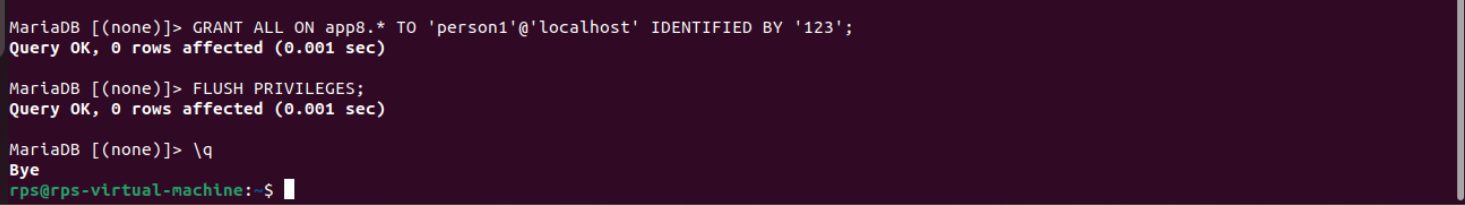
Now, create a database **app8** with the following SQL statement:

*$*CREATE DATABASE app8;

Now, create a new user (let’s say, **person1**), set a password for the user (let’s say **123**) and grant the user permission to use the database **app8**with the following SQL statement:

GRANT ALL ON app8.\* TO 'person1'@'localhost' IDENTIFIED BY '123';

Now, flush the MariaDB privileges for the changes to take effect as follows:

FLUSH PRIVILEGES;

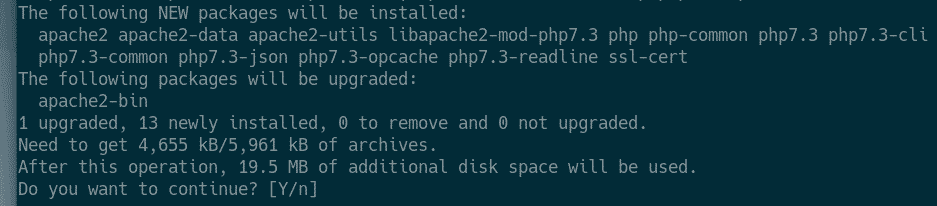
Now, exit out of the MariaDB shell as follows:

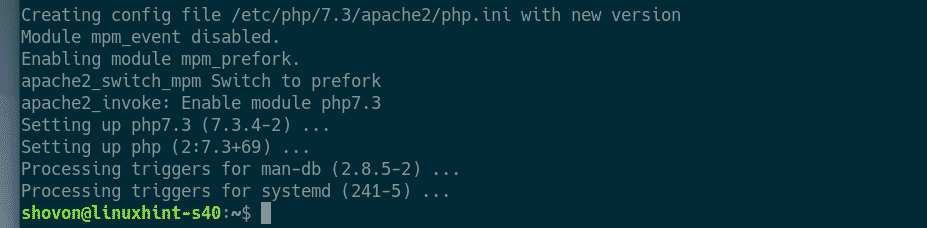
\q

### **Installing Apache Web Server and PHP:**

Now, install Apache 2 web server and PHP with the following command:

*$*sudo apt install apache2 php

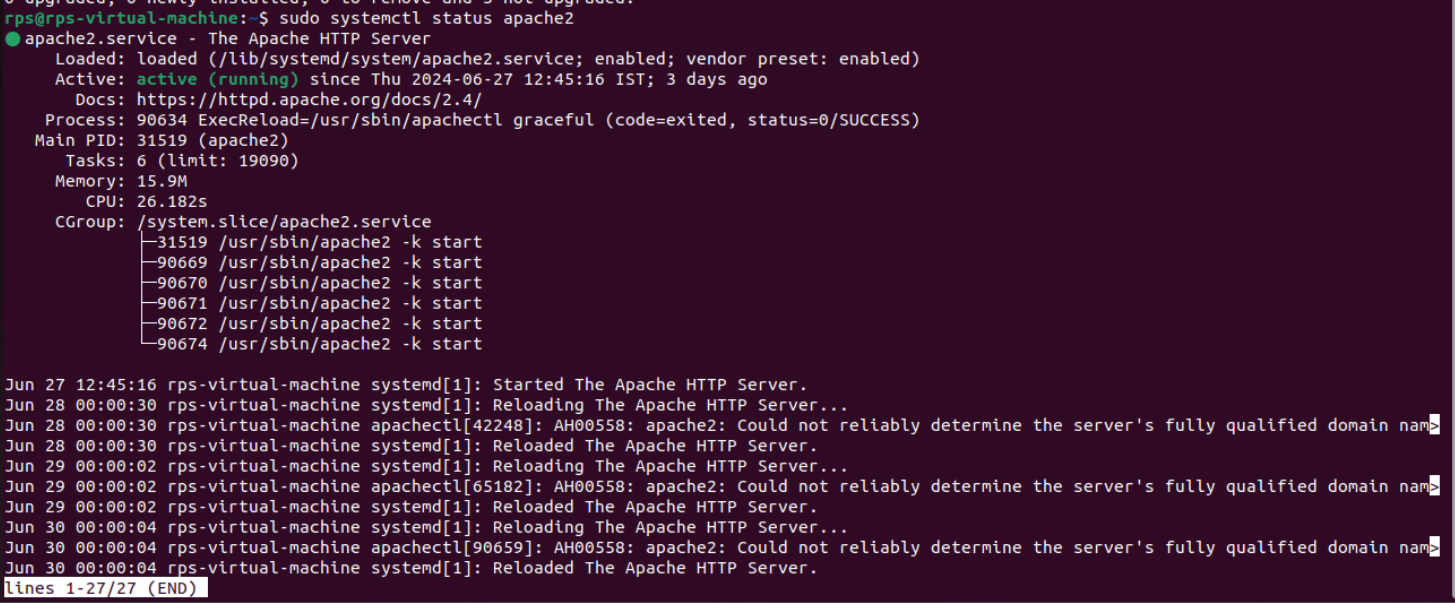
Now, press **Y** and then press **<Enter>** to confirm the installation.

Apache 2 web server should be installed.

Now, check whether **apache2** service is running with the following command:

*$*sudo systemctl status apache2

As you can see, **apache2** service is running. It’s also **enabled** to start automatically on system boot.



If in any case, **apache2** service is not running, then you can start it manually as follows:

$ sudo systemctl start apache2

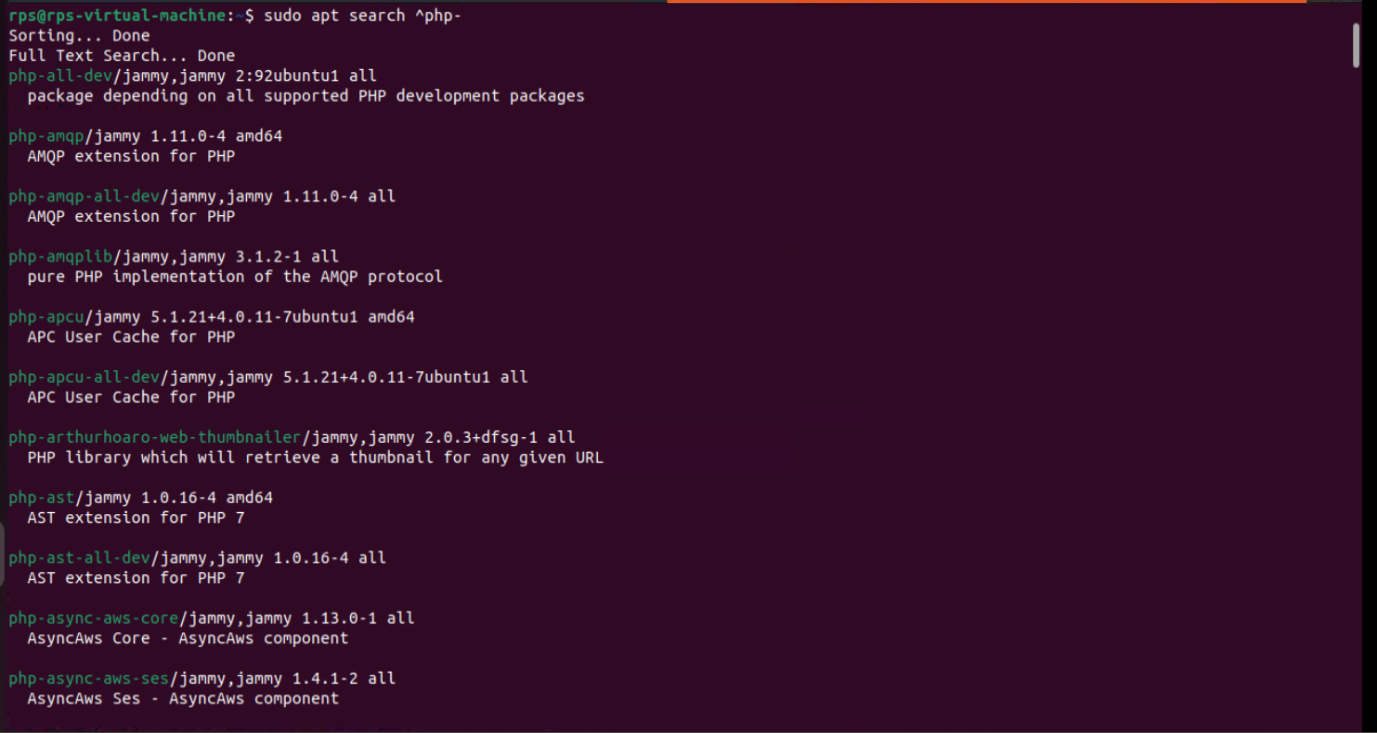
**Installing PHP Extensions:**

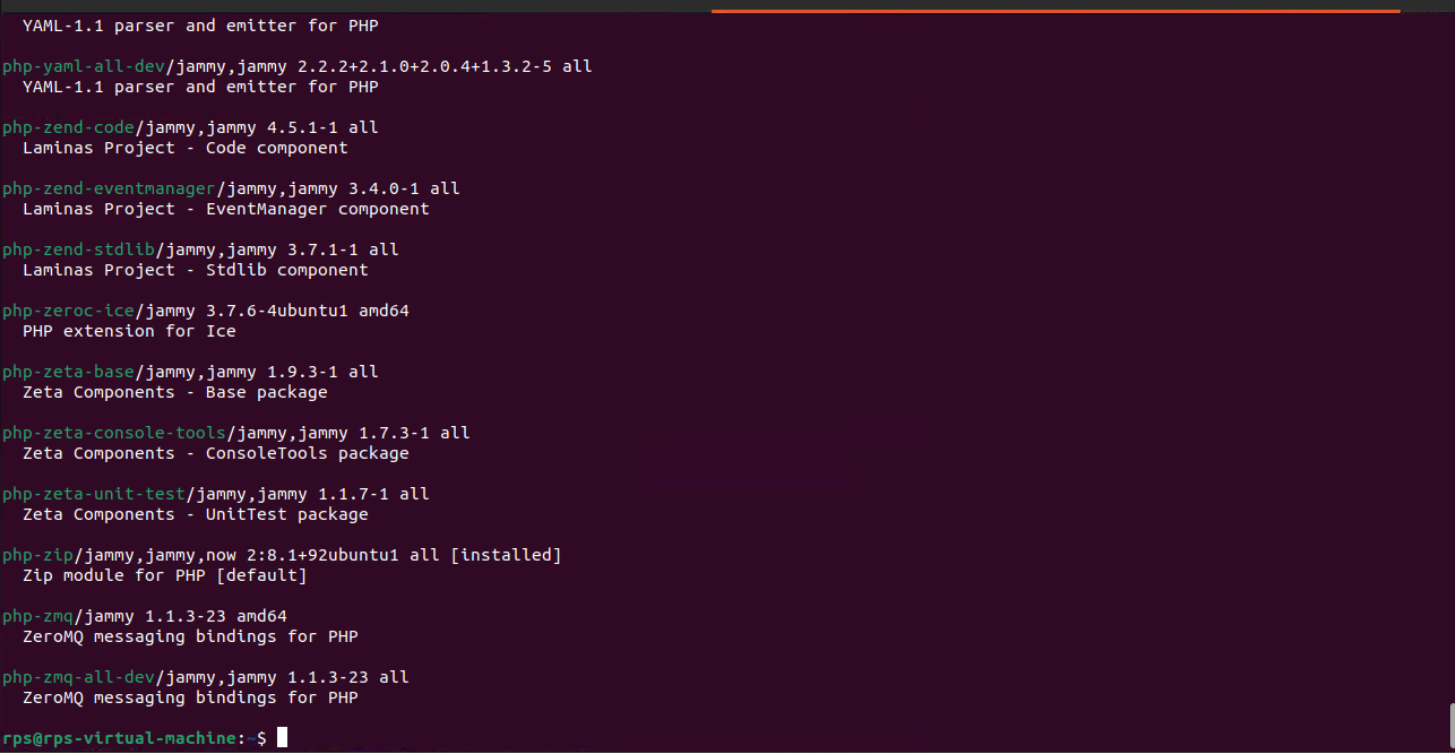
Official package repository has a lot of PHP extensions pre-packaged.

You can list all the available PHP extensions/libraries with the following command:

$ sudo apt search ^php

The package name of all the PHP extensions including their version number and short description should be listed. It’s a very long list. So, it may take a while to find what you’re looking for this way

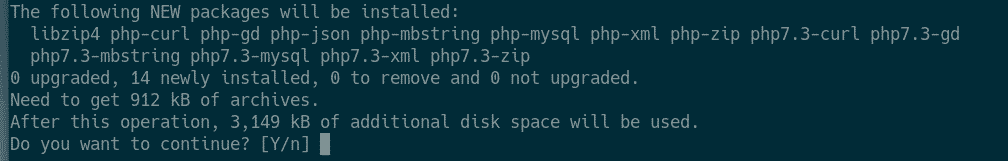




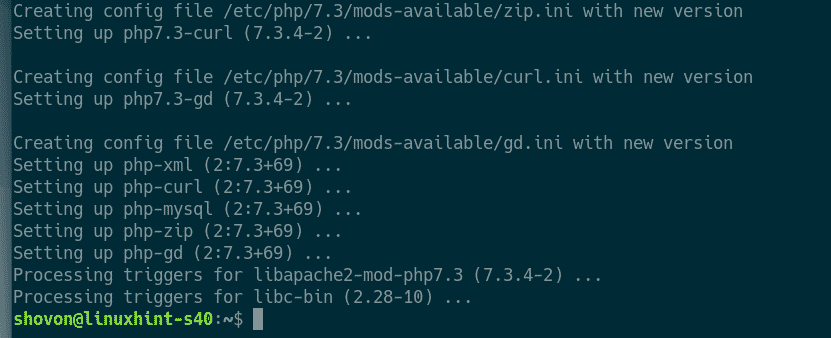
To install the most common PHP extensions/libraries, run the following command:

$ sudo apt install php-curl php-gd php-mbstring php-mysql php-zip php-json php-xml

Now, press **Y** and then press **<Enter>** to confirm the installation.



The common PHP extensions should be installed.



Once the PHP extensions are installed, restart the Apache web server as follows:

*$*sudo systemctl restart apache2

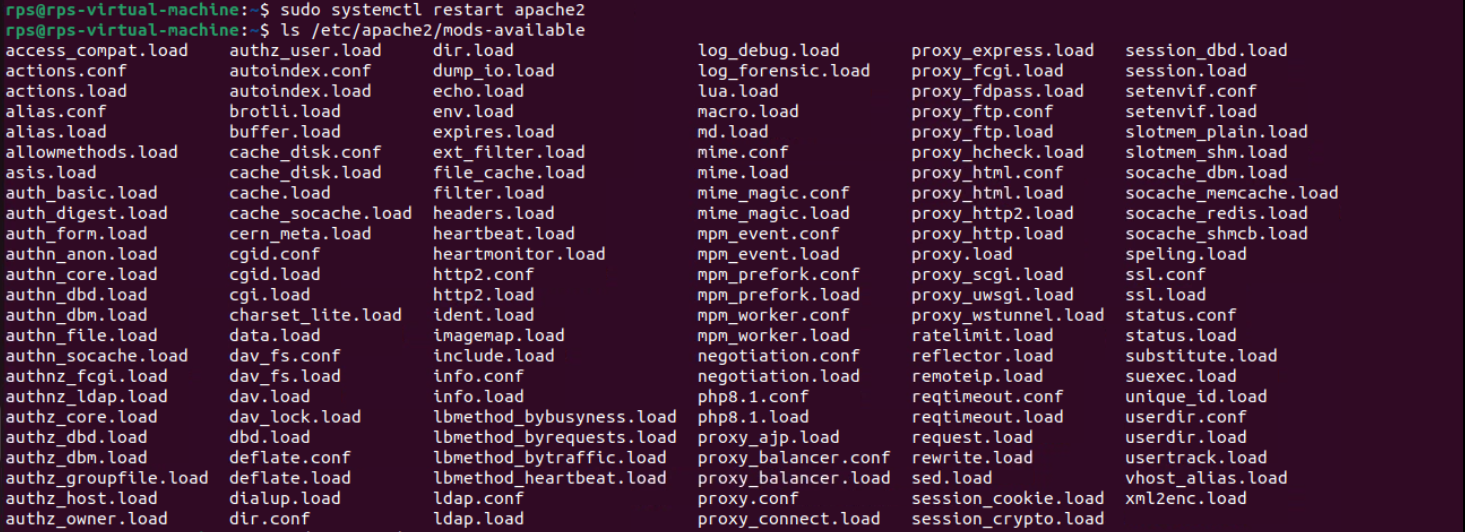
### **Enabling/Disabling Apache Modules:**

Apache web server comes with a lot of modules. You can enable or disable them as you need.

To list all the available Apache 2 modules, run the following command

*$*ls /etc/apache2/mods-available

As you can see, all the available Apache 2 modules are listed.



To enable a module (let’s say, **rewrite**), run the following command:

*$*sudo a2enmod rewrite

Don’t forget to restart the Apache 2 web server if you enable/disable Apache 2 modules.

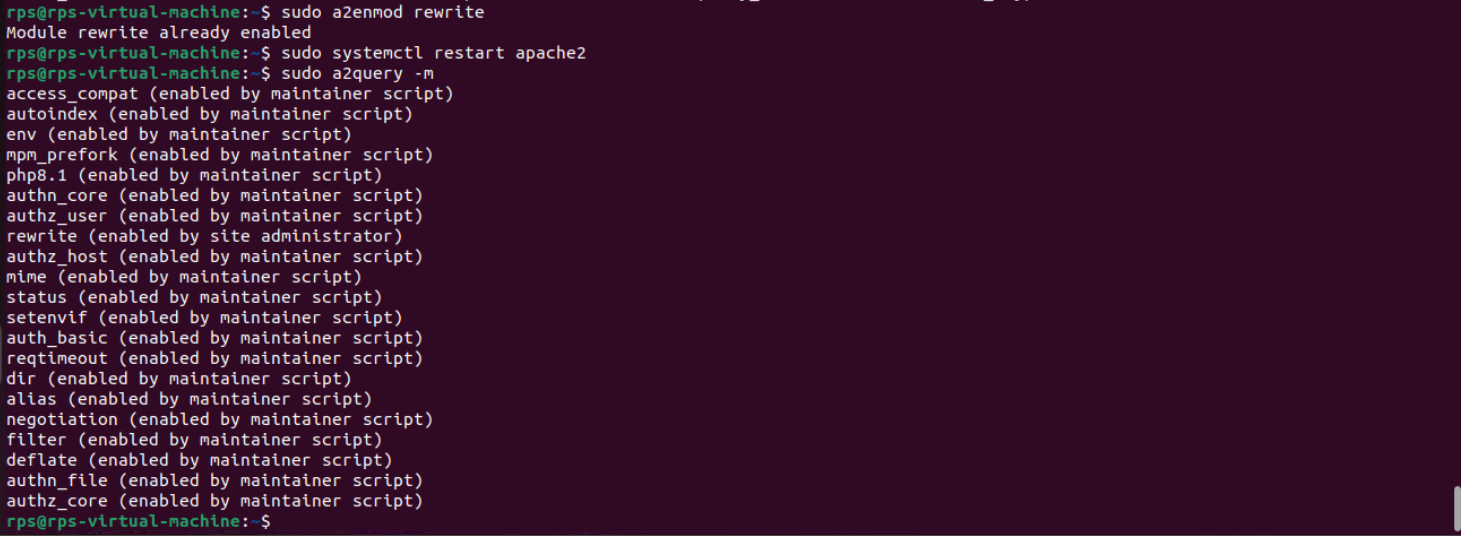
To restart the Apache 2 web server, run the following command:

*$*sudo systemctl restart apache2

To list all the enabled/active Apache 2 modules, run the following command:

*$*sudo a2query -m

All the enabled/active Apache 2 modules should be listed.



If you decide to disable an enabled module, use the **a2dismod** command as follows:

*$*sudo a2dismod moduleName

### **Changing Apache Run User:**

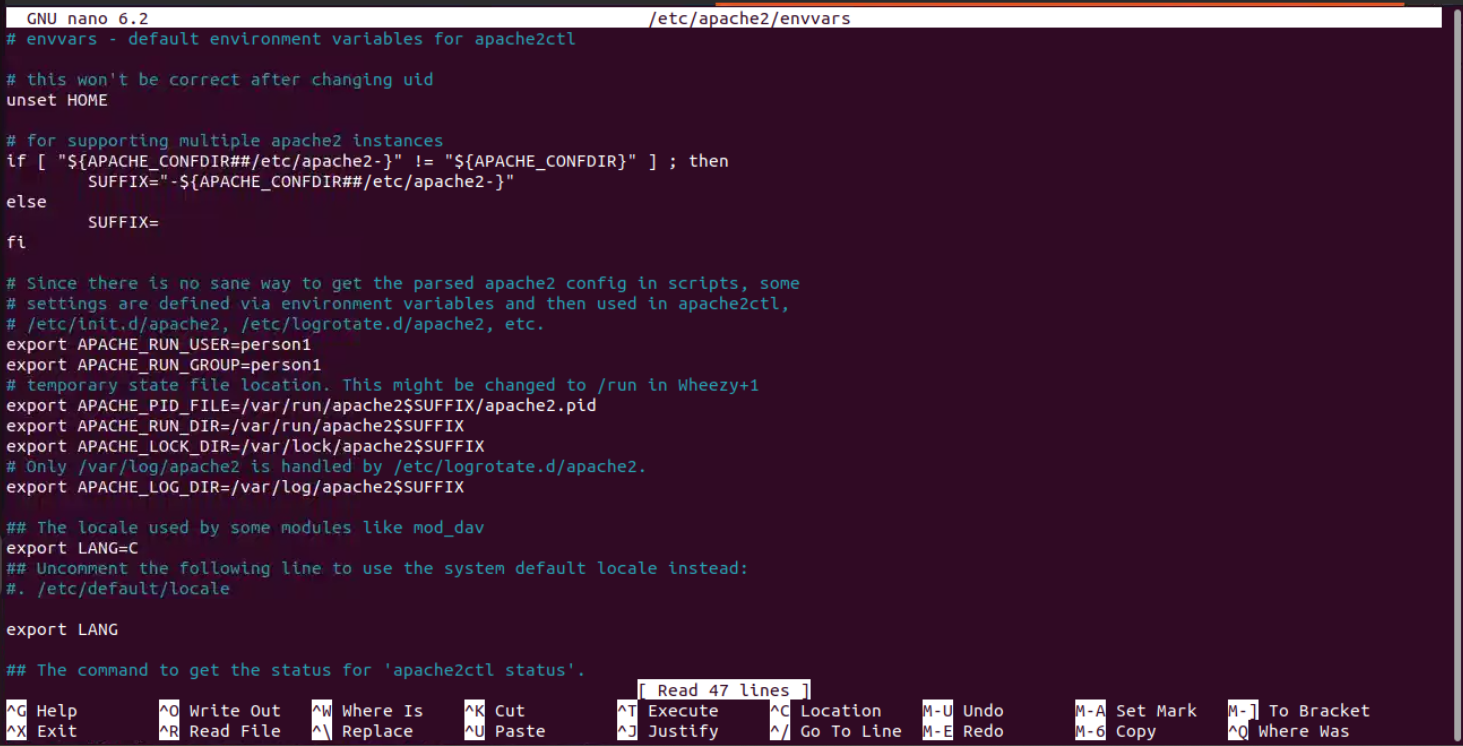
The default Apache run user is **www-data** and the default web root directory is **/var/www/html**. So, as an ordinary user, you won’t be able to create files/directories, or modify existing files/directories in the web root directory. As you’re setting up a development LAMP server, this is not what you want. To solve this problem, you should change the Apache run user to your login user and change the owner and group of the webroot **/var/www/html** to your login user.

To change the Apache run user, edit **/etc/apache2/envvars** configuration file with the following command:

*$*sudo nano /etc/apache2/envvars

You have to modify the **APACHE\_RUN\_USER and APACHE\_RUN\_GROUP environment variables.**

Now, set **APACHE\_RUN\_USER** and **APACHE\_RUN\_GROUP** environment variables to your login user’s username (the output of the **whoami** command). Once you’re done, save the file by pressing **<Ctrl>** + **X** followed by **Y** and **<Enter>**.



Now, change the owner and group of the **/var/www/html**directory to the username of your login user with the following command

*$*sudo chown -Rf $(whoami):$(whoami) /var/www/html

Now, restart the Apache 2 web server with the following command:

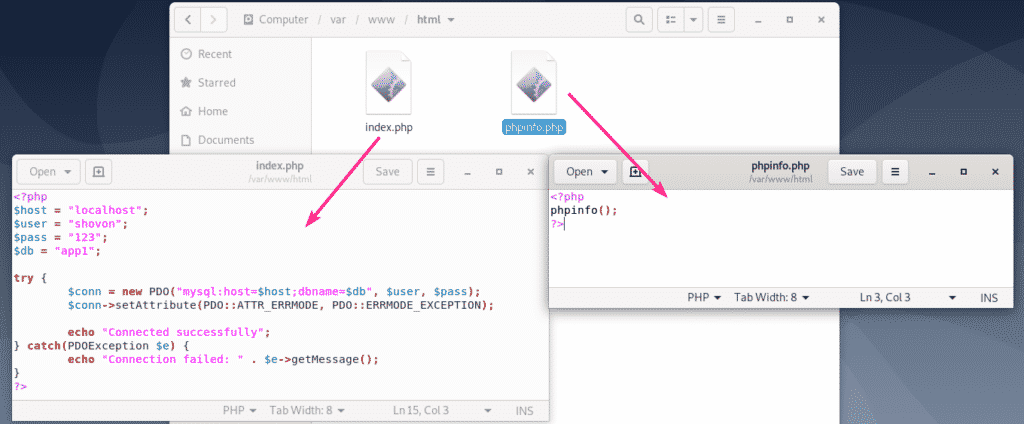
*$*sudo systemctl restart apache2

### **Testing LAMP Server:**

### Create 2 PHP scripts **index.php** and **phpinfo.php** in the webroot **/var/www/html**.

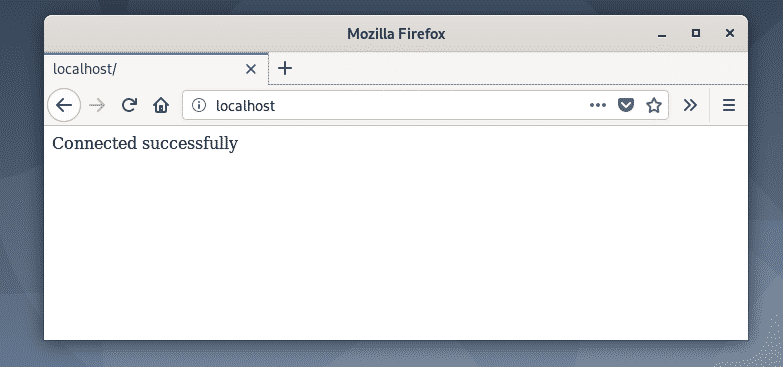
Contents of **index.php**:

<?php  
$host = "localhost";  
$user = "person1";  
$pass = "123";  
$db = "app8";  
   
try {  
$conn = new PDO("mysql:host=$host;dbname=$db", $user, $pass);  
$conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  
   
echo "Connected successfully";  
} catch(PDOException $e) {  
echo "Connection failed: " . $e->getMessage();  
}  
?>  
   
   
Contents of phpinfo.php:  
<?php  
[phpinfo](http://www.php.net/phpinfo)();  
?>

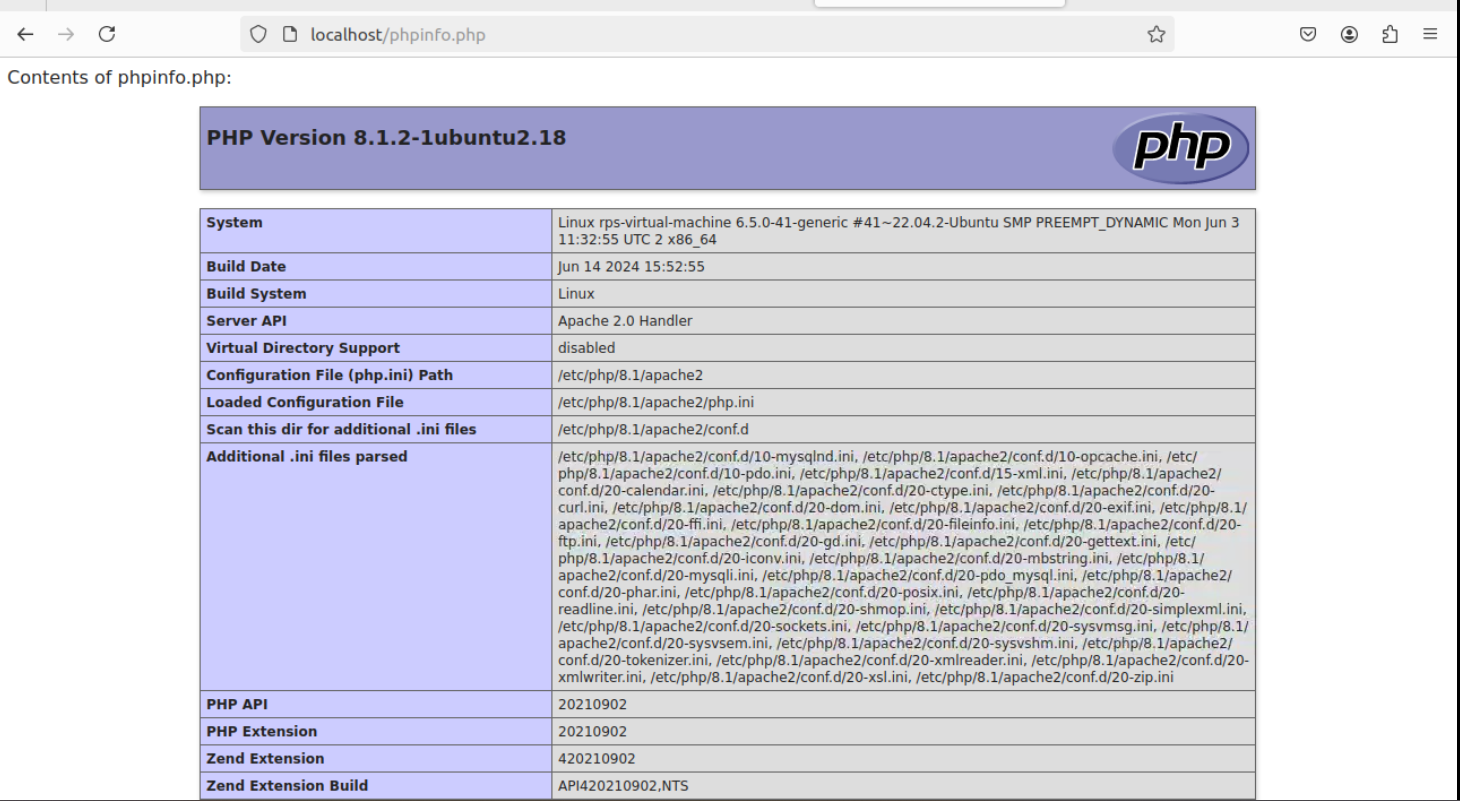


Now, you should be able to access the PHP scripts from your browser as you can see in the screenshot below.

http://localhost



<http://localhost/phpinfo.php>



So, that’s how you setup a LAMP server for PHP web development.

**Conclusion:**

Working on the Linux Web Server project with Mariadb involves developing server-side logic, handling database interactions, and creating secure, efficient APIs. Key responsibilities include writing optimized SQL queries, implementing secure coding practices to prevent vulnerabilities, and integrating PHP with frontend components.