Install Linux, Apache, MySQL, PHP (LAMP) Stack on Ubuntu 22.04

Step 1 — Installing Apache and Updating the Firewall

The Apache web server is among the most popular web servers in the world. It's well documented, has an active community of users, and has been in wide use for much of the history of the web, which makes it a great choice for hosting a website.

Start by updating the package manager cache. If this is the first time you're using `sudo` within this session, you'll be prompted to provide your user's password to confirm you have the right privileges to manage system packages with `apt`:

```
sudo apt update
```

Then, install Apache with:

```
sudo apt install apache2
```

You'll be prompted to confirm Apache's installation. Confirm by pressing `Y`, then `ENTER`.

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
```

```
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"
```

Verify the change with:

```
sudo ufw status
```

```
Output
```

```
Status: active

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

Apache (v6)                ALLOW       Anywhere (v6)
```

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

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.

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
```

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.

Warning: As of July 2022, an error will occur when you run the `mysql_secure_installation` script without some further configuration. The reason is that

this script will attempt to set a password for the installation's root MySQL account but, by default on Ubuntu installations, this account is not configured to connect using a password.

Prior to July 2022, this script would silently fail after attempting to set the root account password and continue on with the rest of the prompts. However, as of this writing the script will return the following error after you enter and confirm a password:

```
Output
```

```
... Failed! Error: SET PASSWORD has no significance for user
'root'@'localhost' as the authentication method used doesn't store
authentication data in the MySQL server. Please consider using ALTER USER
instead if you want to change authentication parameters.

New password:
```

This will lead the script into a recursive loop which you can only get out of by closing your terminal window.

Because the `mysql_secure_installation` script performs a number of other actions that are useful for keeping your MySQL installation secure, it's still recommended that you run it before you begin using MySQL to manage your data. To avoid entering this recursive loop, though, you'll need to first adjust how your root MySQL user authenticates.

First, open up the MySQL prompt:

```
sudo mysql
```

Then run the following `ALTER USER` command to change the root user's authentication method to one that uses a password. The following example changes the authentication method to `mysql_native_password`:

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

After making this change, exit the MySQL prompt:

```
exit
```

Following that, you can run the `mysql_secure_installation` script without issue.

Start the interactive script by running:

```
sudo mysql_secure_installation
```

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:

```
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 whether you're able to log in to the MySQL console by typing:

```
sudo mysql
```

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. Below is an example output:

```
Output
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.28-0ubuntu4 (Ubuntu)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

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:

```
exit
```

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 seem like a security concern,

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.

Note: There are some older versions of PHP that doesn't support `caching_sha2_password`, the default authentication method for MySQL 8. For that reason, when creating database users for PHP applications on MySQL 8, you may need to configure your application to use the `mysql_native_password` plug-in instead. This tutorial will demonstrate how to do that in Step 6.

Your MySQL server is now installed and secured. Next, you'll install PHP, the final component in the LAMP stack.

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 the following command:

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

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

```
php -v
```

```
Output
```

```
PHP 8.1.2 (cli) (built: Mar  4 2022 18:13:46) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.1.2, Copyright (c) Zend Technologies
    with Zend OPcache v8.1.2, Copyright (c), by Zend Technologies
```

At this point, your LAMP stack is fully operational, but before testing 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.

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.

Apache on Ubuntu 22.04 has one virtual host 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
```

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
```

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
```

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

/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`, 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 pound sign (`#`) the beginning of each option's lines.

Now, use `a2ensite` to enable the new virtual host:

```
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
```

Finally, reload Apache so these changes take effect:

```
sudo systemctl reload apache2
```

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 to test that the virtual host works as expected:

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

Include the following content in this file:

/var/www/your_domain/index.html

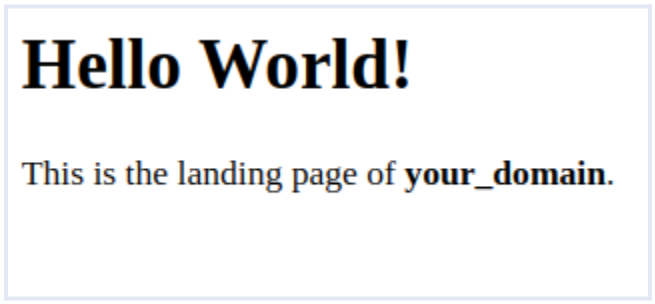
```
<html>
  <head>
    <title>your_domain website</title>
  </head>
  <body>
    <h1>Hello World!</h1>
```

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

Save and close the file, then go to your browser and access your server's domain name or IP address:

```
http://server_domain_or_IP
```

Your web page should reflect the contents in the file you just edited:



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
```

/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
```

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, 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
```

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

/var/www/your_domain/info.php

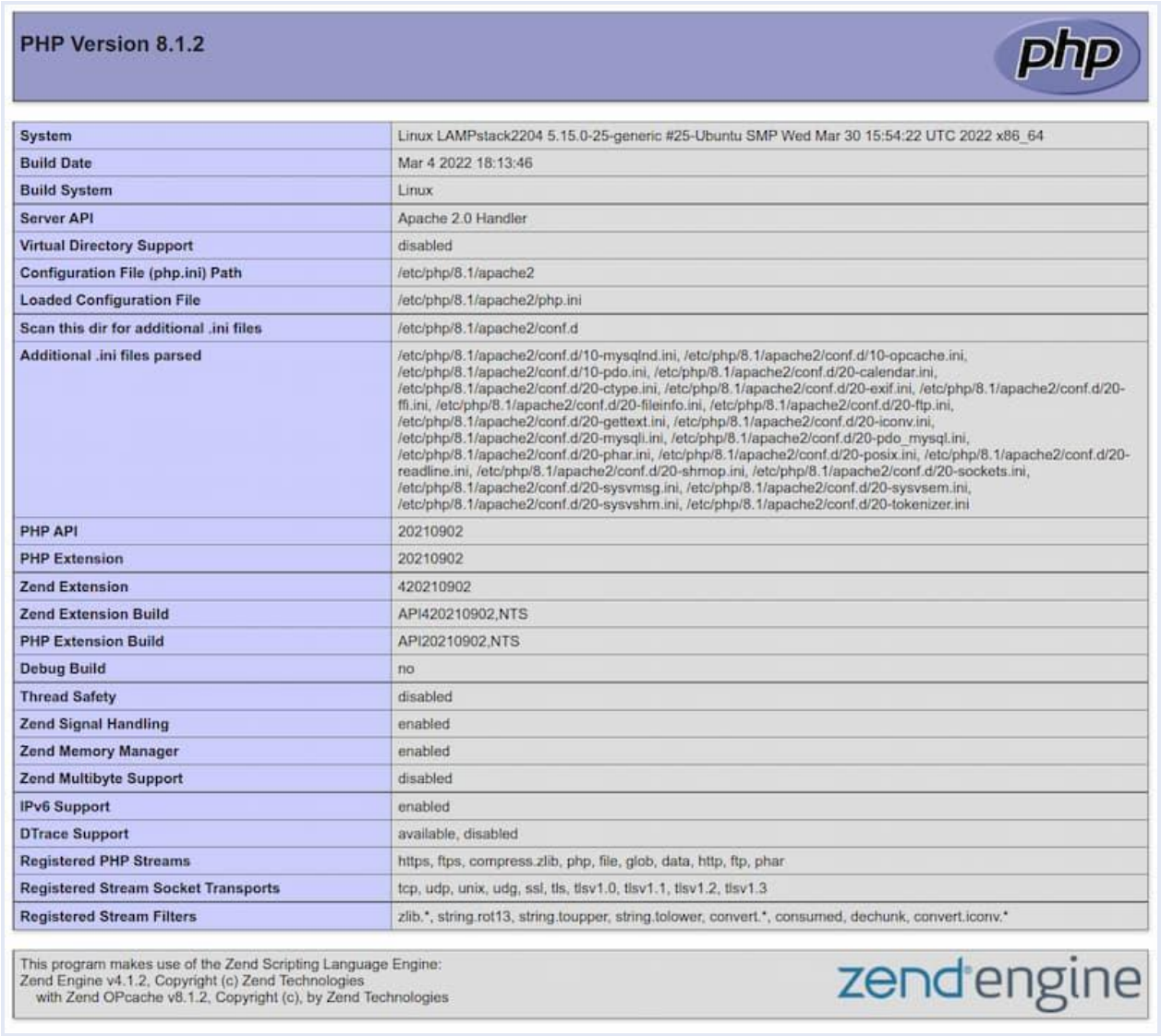
```
<?php
phpinfo();
```

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
```

Here is an example of the default PHP web page:



PHP Version 8.1.2

System	Linux LAMPstack2204 5.15.0-25-generic #25-Ubuntu SMP Wed Mar 30 15:54:22 UTC 2022 x86_64
Build Date	Mar 4 2022 18:13:46
Build System	Linux
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/8.1/apache2
Loaded Configuration File	/etc/php/8.1/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/8.1/apache2/conf.d
Additional .ini files parsed	/etc/php/8.1/apache2/conf.d/10-mysqlnd.ini, /etc/php/8.1/apache2/conf.d/10-opcache.ini, /etc/php/8.1/apache2/conf.d/10-pdo.ini, /etc/php/8.1/apache2/conf.d/20-calendar.ini, /etc/php/8.1/apache2/conf.d/20-ctype.ini, /etc/php/8.1/apache2/conf.d/20-exif.ini, /etc/php/8.1/apache2/conf.d/20-ffi.ini, /etc/php/8.1/apache2/conf.d/20-fileinfo.ini, /etc/php/8.1/apache2/conf.d/20-ftp.ini, /etc/php/8.1/apache2/conf.d/20-gettext.ini, /etc/php/8.1/apache2/conf.d/20-iconv.ini, /etc/php/8.1/apache2/conf.d/20-mysqli.ini, /etc/php/8.1/apache2/conf.d/20-pdo_mysql.ini, /etc/php/8.1/apache2/conf.d/20-phar.ini, /etc/php/8.1/apache2/conf.d/20-posix.ini, /etc/php/8.1/apache2/conf.d/20-readline.ini, /etc/php/8.1/apache2/conf.d/20-shmop.ini, /etc/php/8.1/apache2/conf.d/20-sockets.ini, /etc/php/8.1/apache2/conf.d/20-sysvmsg.ini, /etc/php/8.1/apache2/conf.d/20-sysvsem.ini, /etc/php/8.1/apache2/conf.d/20-sysvshm.ini, /etc/php/8.1/apache2/conf.d/20-tokenizer.ini
PHP API	20210902
PHP Extension	20210902
Zend Extension	420210902
Zend Extension Build	API420210902,NTS
PHP Extension Build	API20210902,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
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, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v4.1.2, Copyright (c) Zend Technologies
with Zend OPcache v8.1.2, 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 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. Use `rm` to do so:

```
sudo rm /var/www/your_domain/info.php
```

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

Step 6 — Testing Database Connection from PHP

If you want to test whether PHP is able to connect to MySQL and execute database queries, you can create a test table with test data and query for its contents from a PHP script. Before you do that, you need to create a test database and a new MySQL user properly configured to access it.

Create a database named example_database and a user named example_user. You can replace these names with different values.

First, connect to the MySQL console using the root account:

```
sudo mysql
```

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

```
CREATE DATABASE example_database;
```

Now 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` that authenticates with the `caching_sha2_password` method. We're defining this user's password as `password`, but you should replace this value with a secure password of your own choosing.

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

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:

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

Now give this user permission over the `example_database` database:

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

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:

```
exit
```

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
```

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:

```
SHOW DATABASES;
```

This will give you the following output:

```
Output
```

```
+--------------------+
| Database           |
+--------------------+
| example_database   |
| information_schema |
+--------------------+
2 rows in set (0.000 sec)
```

Next, create a test table named todo_list. From the MySQL console, run the following statement:

```
CREATE TABLE example_database.todo_list (
    
    item_id INT AUTO_INCREMENT,
    
    content VARCHAR(255),
    
    PRIMARY KEY(item_id)
    
);
```

Insert a few rows of content in the test table. Repeat the next command a few times, using different values, to populate your test table:

```
INSERT INTO example_database.todo_list (content) VALUES ("My first important item");
```

To confirm that the data was successfully saved to your table, run:

```
SELECT * FROM example_database.todo_list;
```

The following is the output:

```
Output
```

```
+---------+--------------------------+
| item_id | content                  |
+---------+--------------------------+
|       1 | My first important item  |
|       2 | My second important item |
|       3 | My third important item  |
|       4 | and this one more thing  |
+---------+--------------------------+
4 rows in set (0.000 sec)
```

After confirming that you have valid data in your test table, exit the MySQL console:

```
exit
```

Now you can create the PHP script that will connect to MySQL and query for your content. Create a new PHP file in your custom web root directory using your preferred editor:

```
nano /var/www/your_domain/todo_list.php
```

The following PHP script connects to the MySQL database and queries for the content of the todo_list table, exhibiting the results in a list. If there's a problem with the database connection, it will throw an exception.

Add this content into your `todo_list.php` script, remembering to replace the `example_user` and `password` with your own:

/var/www/your_domain/todo_list.php

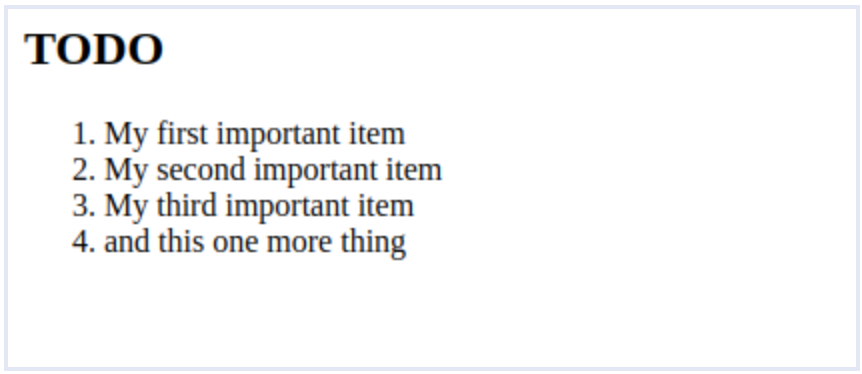
```
<?php
$user = "example_user";
$password = "password";
$database = "example_database";
$table = "todo_list";

try {
  $db = new PDO("mysql:host=localhost;dbname=$database", $user,
$password);
  echo "<h2>TODO</h2><ol>";
  foreach($db->query("SELECT content FROM $table") as $row) {
    echo "<li>" . $row['content'] . "</li>";
  }
  echo "</ol>";
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
```

Save and close the file when you're done editing.

You can now access this page in your web browser by visiting the domain name or public IP address configured for your website, followed by `/todo_list.php`:

```
http://your_domain_or_IP/todo_list.php
```

This web page should reveal the content you've inserted in your test table to your visitor:



That means your PHP environment is ready to connect and interact with your MySQL server.