CONTENTS

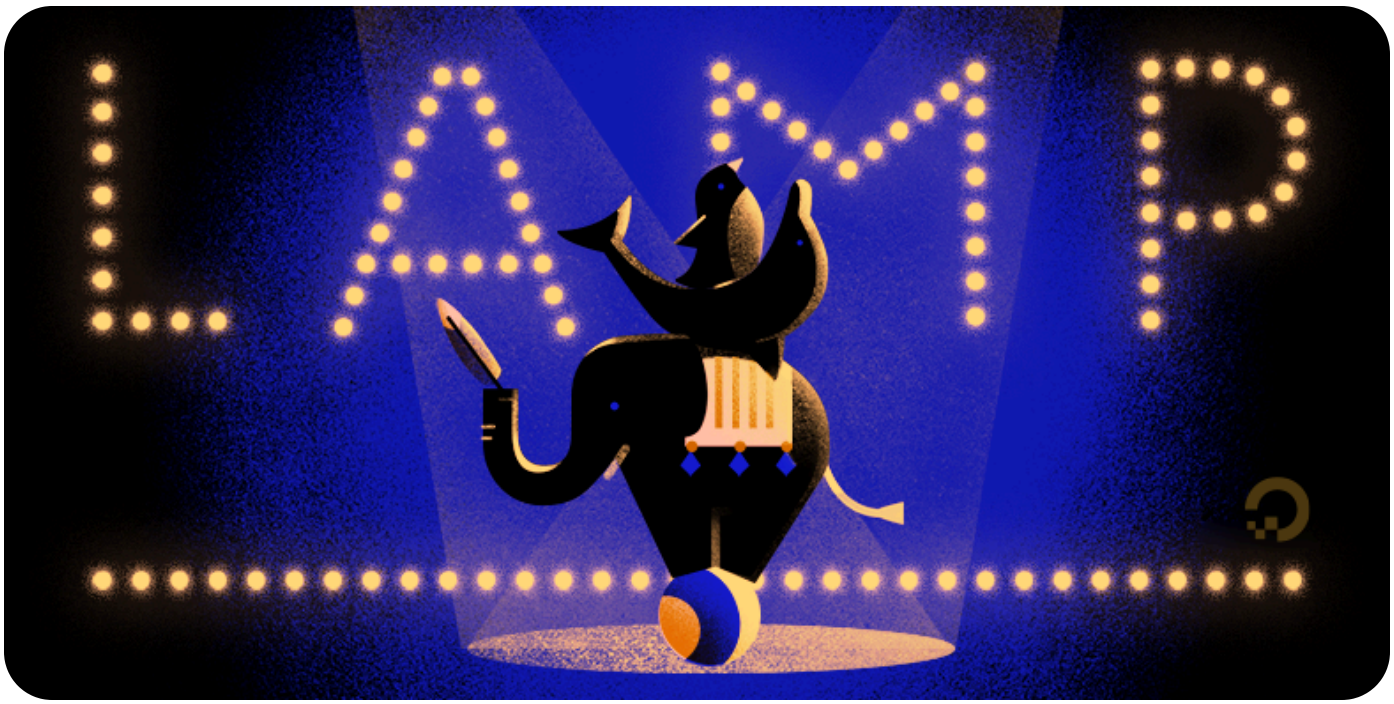
// TUTORIAL //

How To Install Linux, Apache, MariaDB, PHP (LAMP) stack on Debian 10

Updated on January 24, 2023

Apache Debian 10 LAMP Stack

Mark Drake and Erika Heidi



Introduction

A *LAMP* stack is a group of open-source software typically installed together to enable a server to host dynamic websites and web apps. This term is an acronym that represents the **L**inux operating system, with the **A**pache web server. The site data is stored in a **M**ariaDB database, and dynamic content is processed by **P**HP.

Although this software stack typically includes **MySQL** as the database management system, some Linux distributions — including Debian — use MariaDB as a drop-in replacement for MySQL.

In this guide, you will install a LAMP stack on a Debian 10 server, using MariaDB as the database management system.

Prerequisites

To follow this tutorial, you will need to have a Debian 10 server with a non-**root** `sudo`-enabled user account and a basic firewall. This can be configured using our initial server setup guide for Debian 10.

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 and has been in wide use for much of the history of the web, which makes it a great default 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`:

Copy

```
$ sudo apt update
```

Then install Apache with the following:

This command prompts you to confirm Apache's installation. Confirm by pressing `Y`, then `ENTER`. Once the installation is complete, you need to adjust your firewall settings. Assuming that you followed the initial server setup instructions to install and enable the UFW firewall, make sure that your firewall allows HTTP and HTTPS traffic.

On Debian 10, UFW comes loaded with app profiles that you can use to adjust your firewall settings. View the full list of application profiles by running:

Copy

```
$ sudo ufw app list
```

The `WWW` profiles are used to manage ports used by web servers:

```
Output
Available applications:
. . .
  WWW
  WWW Cache
  WWW Full
  WWW Secure
. . .
```

If you inspect the `WWW Full` profile, it shows that it enables traffic to ports `80` and `443`:

Copy

```
$ sudo ufw app info "WWW Full"
```

```
Output
Profile: WWW Full
Title: Web Server (HTTP,HTTPS)
```

Allow incoming HTTP and HTTPS traffic for this profile:

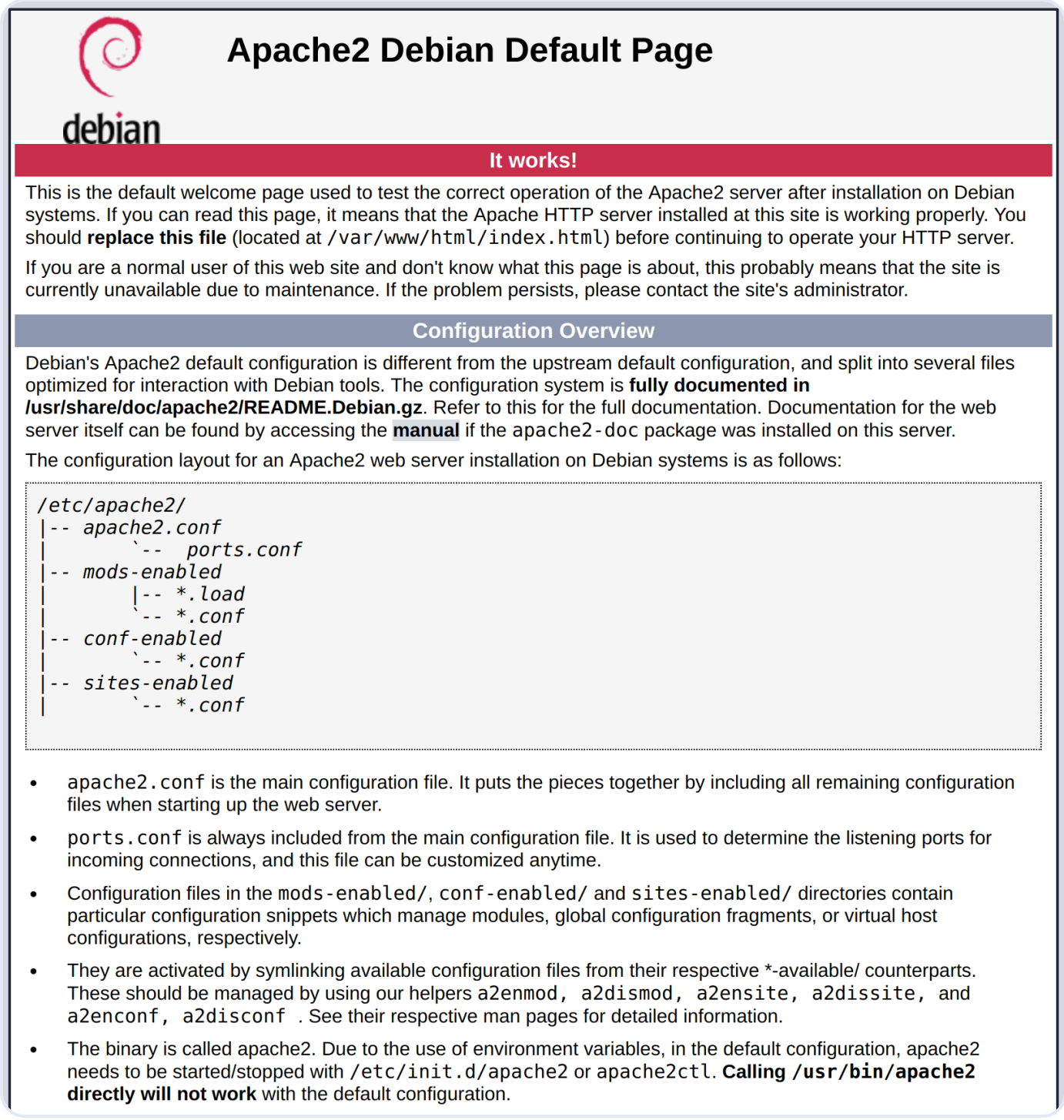
Copy

```
$ sudo ufw allow in "WWW Full"
```

You can verify that everything went as planned by visiting your server's public IP address in your web browser:

```
http:// your_server_ip
```

This will return the default Debian 10 Apache web page, which is there for informational and testing purposes:



Apache2 Debian Default Page

debian

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. 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

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian 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 Debian 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. Due to the use of environment variables, in the default configuration, apache2 needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

If your browser returns this page, then your web server is now correctly installed and accessible through your firewall.

How To Find Your Server's Public IP Address

Copy

```
$ ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'
```

This will return two or three lines back. They are all correct addresses, but your computer may only be able to use one, 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 views your server. You can run the following command and ask a specific server what your IP address is:

Since Debian 10 does not have `curl` be default, you will need to install it first:

Copy

```
$ sudo apt install curl
```

Then run the following command and ask a specific server what your IP address is:

Copy

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

Regardless of the method, write your IP address into your web browser to verify that your server is running the default Apache page.

Step 2 – Installing MariaDB

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.

In Debian 10, the metapackage `mysql-server`, which was traditionally used to install the MySQL server, was replaced by `default-mysql-server`. This metapackage references

For longer-term compatibility, however, it's recommended that instead of using the metapackage you install MariaDB using the program's actual package, `mariadb-server`.

To install the MariaDB software, run:

Copy

```
$ sudo apt install mariadb-server
```

When the installation is finished, it's recommended that you run a security script that comes pre-installed with MariaDB. This script will remove some insecure default settings and lock down access to your database system. Start the interactive script by running:

Copy

```
$ sudo mysql_secure_installation
```

This script will take you through a series of prompts where you can make some changes to your MariaDB setup. The first prompt will ask you to enter the current **database root** password. 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. Since you recently installed MariaDB and haven't made any configuration changes yet, this password will be

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

MariaDB [(none)]>
```

Notice that you didn't need to provide a password to connect as the **root** user. That works because the default authentication method for the administrative MariaDB user is `unix_socket` instead of `password`. Even though this might seem like a security concern at first, it makes the database server more secure because the only users allowed to log in as the **root** MariaDB 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.

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.

You can exit the MariaDB console with the following:

Copy

```
MariaDB [(none)]> exit
```

Your MariaDB server is now installed and secured. Next, you will install PHP, the final component in the LAMP stack.

You have Apache installed to serve your content and MariaDB installed to store and manage your data. PHP is the component of your setup that will process code to display dynamic content to the final user. It can run scripts, connect to your MariaDB databases to get information, and hand the processed content over to your web server to display.

In addition to the `php` package, you will need `php-mysql`, a PHP module that allows PHP to communicate with MySQL-based database, such as MariaDEB. You will 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:

Copy

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

Once installation is complete, you can verify your PHP version with the following command:

Copy

```
$ php -v
```

```
Output
PHP 7.3.31-1~deb10u2 (cli) (built: Dec 15 2022 09:39:10) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.3.31, Copyright (c) 1998-2018 Zend Technologies
    with Zend OPcache v7.3.31-1~deb10u2, Copyright (c) 1999-2018, by Zend Techno
```

In most cases, you will want to modify the way that Apache serves files. Currently, if a user requests a directory from the server, Apache will first search for a file called `index.html`. To instruct the web server to prefer PHP files over others, you can set Apache to search for an `index.php` file first.

To do this, run the following command to open the `dir.conf` file in your preferred text

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

The contents will be as follows:

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

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

Move the PHP index file to the first position after the `DirectoryIndex` specification, like in the following:

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

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

When you are finished, save and close the file. If you're using `nano`, you can do so by pressing `CTRL+X`, then `Y` and `ENTER` to confirm.

Now reload Apache's configuration:

Copy

```
$ sudo systemctl reload apache2
```

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

Copy

```
Sample Output
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset:
   Active: active (running) since Fri 2023-01-20 22:21:24 UTC; 2min 12s ago
     Docs: https://httpd.apache.org/docs/2.4/
  Process: 13076 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCC
  Process: 13097 ExecReload=/usr/sbin/apachectl graceful (code=exited, status=0/
 Main PID: 13080 (apache2)
    Tasks: 6 (limit: 4915)
   Memory: 13.7M
   CGroup: /system.slice/apache2.service
           ├─13080 /usr/sbin/apache2 -k start
           ├─13101 /usr/sbin/apache2 -k start
           ├─13102 /usr/sbin/apache2 -k start
           ├─13103 /usr/sbin/apache2 -k start
           ├─13104 /usr/sbin/apache2 -k start
           └─13105 /usr/sbin/apache2 -k start
```

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. You will set this up 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 section, you'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 a DNS provider, check out our product documentation for detailed instructions on how to set up a new domain name and point it to you server

By default, Apache serves its content from a directory located at `/var/www/html`, using the configuration contained in `/etc/apache2/sites-available/000-default.conf`. Instead of modifying the default website configuration file, you are going to create a new virtual host

Begin by creating the root web directory for **your_domain** as follows:

Copy

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

Next, assign ownership of the directory with the `$USER` environment variable, which will reference your current system user:

Copy

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

Then, open a new configuration file in Apache's `sites-available` directory using your preferred text editor. `nano` is used in the following example:

Copy

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

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

/etc/apache2/sites-available/your_domain

```
<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>
```

domain name, you can remove or comment out the options `ServerName` and `ServerAlias` by adding a pound sign (`#`) character in the beginning of each option's lines.

Save and close the file when you're done.

Now use `a2ensite` to enable this virtual host:

Copy

```
$ 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 overwrite your Virtual Host. To disable Apache's default website, run:

Copy

```
$ sudo a2dissite 000-default
```

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

Copy

```
$ sudo apache2ctl configtest
```

Finally, reload Apache so these changes take effect:

Copy

```
$ sudo systemctl reload apache2
```

Next, you will create a PHP script to test that PHP is correctly installed and configured on

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.

Start by creating a new file named `info.php` inside your custom web root folder:

Copy

```
$ 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 the 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:// your_domain /info.php
```

Here is an example of the default PHP web page:

PHP Version 7.3.31-1~deb10u2

System	Linux lamp 4.19.0-20-cloud-amd64 #1 SMP Debian 4.19.235-1 (2022-03-17) x86_64
Build Date	Dec 15 2022 09:39:10
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.3/apache2
Loaded Configuration File	/etc/php/7.3/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.3/apache2/conf.d
Additional .ini files parsed	/etc/php/7.3/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.3/apache2/conf.d/10-opcache.ini, /etc/php/7.3/apache2/conf.d/10-pdo.ini, /etc/php/7.3/apache2/conf.d/20-calendar.ini, /etc/php/7.3/apache2/conf.d/20-ctype.ini, /etc/php/7.3/apache2/conf.d/20-exif.ini, /etc/php/7.3/apache2/conf.d/20-fileinfo.ini, /etc/php/7.3/apache2/conf.d/20-ftp.ini, /etc/php/7.3/apache2/conf.d/20-gettext.ini, /etc/php/7.3/apache2/conf.d/20-iconv.ini, /etc/php/7.3/apache2/conf.d/20-json.ini, /etc/php/7.3/apache2/conf.d/20-mysqli.ini, /etc/php/7.3/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.3/apache2/conf.d/20-phar.ini, /etc/php/7.3/apache2/conf.d/20-posix.ini, /etc/php/7.3/apache2/conf.d/20-readline.ini, /etc/php/7.3/apache2/conf.d/20-shmop.ini, /etc/php/7.3/apache2/conf.d/20-sockets.ini, /etc/php/7.3/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.3/apache2/conf.d/20-sysvsem.ini, /etc/php/7.3/apache2/conf.d/20-sysvshm.ini, /etc/php/7.3/apache2/conf.d/20-tokenizer.ini
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,NTS
PHP Extension Build	API20180731,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
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 v3.3.31, Copyright (c) 1998-2018 Zend Technologies
with Zend OPcache v7.3.31-1~deb10u2, Copyright (c) 1999-2018, by Zend Technologies

This page provides some basic 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 receive 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 Debian server. You can use `rm` to do so:

Copy

Step 6 – Testing Database Connection from PHP (Optional)

If you want to test whether PHP is able to connect to MariaDB 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 database and a new MariaDB user properly configured to access it.

First, connect to the MariaDB console using the **root** account:

Copy

```
$ sudo mariadb
```

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

Copy

```
MariaDB [(none)]> CREATE DATABASE  example_database ;
```

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

The following command creates a new user named `example_user` that authenticates with a password. We're defining this user's password as `password` , but you should replace this value with a secure password of your own choosing:

Copy

```
MariaDB [(none)]> CREATE USER ' example_user '@'%' IDENTIFIED BY ' password ';
```

Next, give this user permission over the `example_database` database:

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.

Next, flush the privileges to ensure that they are saved and available in the current session:

Copy

```
MariaDB [(none)]> FLUSH PRIVILEGES;
```

Following this, exit the MariaDB shell:

Copy

```
MariaDB [(none)]> exit
```

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

Copy

```
$ mariadb -u  example_user  -p
```

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

Copy

```
MariaDB [(none)]> SHOW DATABASES;
```

This will give you the following output:

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

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

Copy

```
MariaDB [(none)]> CREATE TABLE  example_database . todo_list  (
MariaDB [(none)]>       item_id INT AUTO_INCREMENT,
MariaDB [(none)]>       content VARCHAR(255),
MariaDB [(none)]>       PRIMARY KEY(item_id)
MariaDB [(none)]> );
```

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:

Copy

```
MariaDB [(none)]> INSERT INTO  example_database . todo_list  (content) VALUES ("
```

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

Copy

```
MariaDB [(none)]> SELECT * FROM  example_database . todo_list ;
```

You will receive the following output:

```
Output
+---------+-------------------------+
| item_id | content                 |
```

```
+---------+--------------------------+
4 rows in set (0.000 sec)
```

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

Copy

```
MariaDB [(none)]> exit
```

Now you can create the PHP script that will connect to MariaDB and query for your content. Create a new PHP file in your custom web root directory using your preferred editor. `nano` is used in this example:

Copy

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

The following PHP script connects to the MariaDB 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` values with your own:

/var/www/your_domain/todo_list.php Copy

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

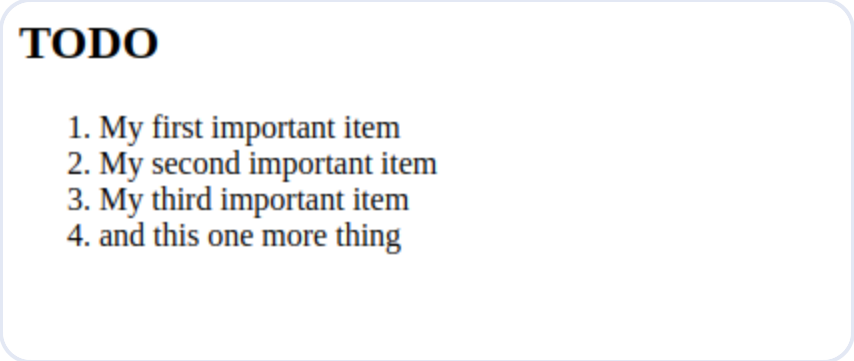
```
    echo "</ol>";
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
```

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 for your website, followed by `/todo_list.php`:

```
http:// your_domain /todo_list.php
```

This web page will 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 MariaDB server.

Conclusion

In this guide, you've built a flexible foundation for serving PHP websites and applications to your visitors, using Apache as a web server and MariaDB as the database system.

As an immediate next step, you should ensure the connections to your web server are secured, by serving them via HTTPS. To accomplish that, you can use Let's Encrypt. You can also read our guide on How to Install and Use Composer for dependency and package management in PHP.

Learn more about our products →

About the authors

Mark Drake Author

Manager, Developer Education

Technical Writer @ DigitalOcean

Erika Heidi Author

Developer Advocate

Dev/Ops passionate about open source, PHP, and Linux.

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

Leave a comment...

This textbox defaults to using **Markdown** to format your answer.

You can type **!ref** in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

najibelazoui • August 7, 2021

hello there i use those insructions but you forget to change in host file

sudo nano hosts **** ADD THIS LINE *** 127.0.0.1 your_domain

Reply

pcnerd19 • February 29, 2020

On the todo list test page, it just says “TODO” without displaying anything from the database.

Show replies Reply

rphintze • December 21, 2019

I am working thru this tutorial, but have run into a problem with the php installation.

When I run:

I get the following response:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package php
E: Unable to locate package libapache2-mod-php
E: Unable to locate package php-mysql
```

Any suggestions?

Show replies Reply

rulet • November 22, 2019

I figured out that it is not depends from ufw, because without ufw Apache sometimes works and sometimes not. I cannot get what is the problem. The start page of Apache server is loaded correctly but sudo systemctl status apache2.service command shows that server failed to start, and mariadb base is not loading by browser...

Show replies Reply

rulet • November 22, 2019

The problem repeated. But after I disabled ufw and rebooted system(Debian 10, 64 bit) apache server worked again. So any ideas how to configure ufw properly?

Reply

but after reboot I'm getting this:

```
r@prime:~$ sudo systemctl status apache2.service
[sudo] пароль для r:
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor
   Active: failed (Result: exit-code) since Mon 2019-11-18 19:12:18 EET;
     Docs: https://httpd.apache.org/docs/2.4/
  Process: 723 ExecStart=/usr/sbin/apachectl start (code=exited, status=

ноя 18 19:12:18 prime apachectl[723]: AH00558: apache2: Could not reliab
ноя 18 19:12:18 prime apachectl[723]: (98)Address already in use: AH0007
ноя 18 19:12:18 prime apachectl[723]: (98)Address already in use: AH0007
ноя 18 19:12:18 prime apachectl[723]: no listening sockets available, sh
ноя 18 19:12:18 prime apachectl[723]: AH00015: Unable to open logs
ноя 18 19:12:18 prime apachectl[723]: Action 'start' failed.
ноя 18 19:12:18 prime apachectl[723]: The Apache error log may have more
ноя 18 19:12:18 prime systemd[1]: apache2.service: Control process exite
ноя 18 19:12:18 prime systemd[1]: apache2.service: Failed with result 'e
ноя 18 19:12:18 prime systemd[1]: Failed to start The Apache HTTP Server
r@prime:~$
```

What am I missing?

Show replies Reply

Sign up

Popular Topics

AI/ML

Ubuntu

Linux Basics

JavaScript

Python

MySQL

Docker

Kubernetes

All tutorials →

Talk to an expert →

Become a contributor for community

Get paid to write technical tutorials and select a tech-focused charity to receive a matching donation.

Sign Up →

DigitalOcean Documentation

Full documentation for every DigitalOcean product.

Learn more →

Resources for startups and SMBs

The Wave has everything you need to know about building a business, from raising funding to marketing your product.

Learn more →

Get our newsletter

Stay up to date by signing up for DigitalOcean's Infrastructure as a Newsletter.

Email address Submit

New accounts only. By submitting your email you agree to our Privacy Policy

*This promotional offer applies to new accounts only.

Company

Products

Resources

Solutions

Contact