EngEd Author Bio

Joseph Chege

Joseph Chege is an undergraduate student taking a Bachelor in Business Information Technology, a 4th-year student at Dedan Kimathi University of Technology. Joseph is fluent in Android Mobile Application Development and has a lot of passion for back-end development.

View author's full profile →

PHP Websites using Docker Containers with PHP Apache and MySQL

June 18, 2021

Topics: Containers

Container technology is growing every day. It's a technology that makes application development much easier and faster. It has a clean architecture that ensures application services utilize resources sustainably by dividing an application into smaller services called images. This allows you to set up each service independently without affecting how the other services run.

In this case, Docker provides a docker-compose file that allows you to set all your application environments and run a few commands to fully set up your application in a more elegant and faster approach.

Let's take the case of running a PHP application. You would have to install all environments that you need to run PHP scripts. You need an apache server installed in your server/system and probably a MySQL database. Then set up each environment in a way that will allow you to run your PHP-driven website.

With Docker, things are much more manageable. Docker allows you to set your application with each service running as a microservice. This way, you set a single YML file that will isolate all the services that your application needs to run. The file sets up the PHP Apache server and MySQL database for you. All you need is to specify the parameters that you need your application to run on.

The main advantage that Containers provides, is a scalable environment to run your application services. It ensures that the practices of continuous integration and continuous delivery (CI/CD) pipelines are enhanced across the team. So you only need to share this YML file with every team member. This will set all the necessary environments across the team regardless of the operating system they are running on. Thus team members can synchronize their work without breaking the code.

This guide will show you how we can use Docker development environment to:

- Setup and run a local PHP Apache server instance.
- Serve a dynamic PHP-driven website.
- Setup a MySQL database to run SQL scripts, fetch records, and print them in a PHP-driven website.

We will use the Docker hub images to set up a containerized PHP development environment.

Prerequisites

- Ensure that Docker demons are installed on your computer.
- Basic knowledge of PHP and SQL queries.
- Fundamental understating of how to build and run Docker hub images from a Docker file.
- Understand how containers work.
- Basic knowledge on how to execute Docker and docker-compose commands.

Setting and testing if Docker is running

After you have downloaded and installed the Docker demon, open the Docker engine and make sure the engine is running.

Open a command line and run the following command to verify if Docker is correctly installed on your computer.

```
docker version
```

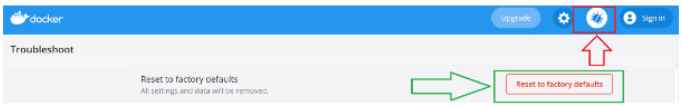
This will log the results almost similar to this command line logs.

```
C:\Users\User>docker version
Client:
 Cloud integration: 1.0.14
 Version:           20.10.6
 API version:       1.41
 Go version:        go1.16.3
 Git commit:        370c289
 Built:             Fri Apr  9 22:49:36 2021
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.6
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.13.15
  Git commit:       8728dd2
  Built:            Fri Apr  9 22:44:56 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.4
  GitCommit:        05f951a3781f4f2c1911b05e61c160e9c30eaa8e
 runc:
  Version:          1.0.0-rc93
  GitCommit:        12644e614e25b05da6fd08a38ffa0cfe1903fdec
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

If you are a beginner at Docker, you might come across this Docker engine error while running the command above.

```
error during connect: This error may indicate that the docker daemon is not running.: Get
 "http://%2F%2F.%2Fpipe%2Fdocker_engine/v1.24/version": open //./pipe/docker_engine: The
system cannot find the file specified.
```

To solve this, head over to your Docker desktop engine, troubleshooting and reset Docker to its factory/defaults setting.



Run the `docker version` command, and it should now work fine. And if the problem persists, search and find the necessary ways to set your Docker engine correctly.

We are starting from scratch; make sure you have no containers and images running in your Docker engine. Run `docker container ls` to check any available container. To remove a Docker container, run `docker container rm <container's name>`. And make sure no container is running.

```
C:\Users\User>docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Laying down docker-compose YML file

Docker-compose allows you to set the parameters of the necessary images that you want to run in your application. In our case, we will use Docker hub official images such as PHP Apache, and MySQL. We will write their parameters in a `.yml` file. A `.yaml` will work as well.

Go ahead and create a project folder and create a `.yml` file inside that folder.

For example, `docker-compose.yml`.

To set a docker-compose, you need first to select the Docker version you want to use, the services you want to provide, and the containers you want to run.

```yaml
version: '3.8'
services:
  php-apache-environment:
    container_name:
```

Setup and run a local PHP Apache server instance

To set up a PHP Apache container, you need to specify the following environments,

- The container name - this is just a random name that you would like to name your PHP container.

For example `container_name: php-apache`.

- The container image - this the official PHP image, the version of PHP Apache you want to use. In this case, we are pulling `image: php:8.0-apache` from the Docker hub.

- The volume - this will set up your present working `src` directory for your code/source files. If you were to run a PHP script, that file would have to be in that directory.

Such as:

```
volumes:
  - ./php/src:/var/www/html/
```

- The port numbers. This defines the ports where the script will run from. It will set up an Apache server port mapping to the port on your local computer.

For example:

```
ports:
  - 8000:80
```

This means that we are setting up an Apache server to expose port 80. Port 8000 reaches out to the PHP scripts and executes them in a browser from within Docker containers.

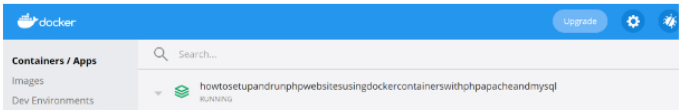
This is how you `docker-compose.yml` should look like.

```
version: '3.8'
services:
  php-apache-environment:
    container_name: php-apache
    image: php:8.0-apache
    volumes:
      - ./php/src:/var/www/html/
    ports:
      - 8000:80
```

Let's test it out. Go ahead and run `docker-compose up`. That's going to pull all the information, download the Apache server, build the image, and run the container.

```
Starting e1fcbc38fde2_php-apache ... done
Attaching to e1fcbc38fde2_php-apache
e1fcbc38fde2_php-apache  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.2. Set the 'ServerName' directive globally to suppress this message
e1fcbc38fde2_php-apache  | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.2. Set the 'ServerName' directive globally to suppress this message
e1fcbc38fde2_php-apache  | [Sat May 22 08:19:21.947726 2021] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.38 (Debian) PHP/8.0.6 configured -- resuming normal operations
e1fcbc38fde2_php-apache  | [Sat May 22 08:19:21.947803 2021] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
```

If you open the Docker desktop engine, the container should be up and running.



To ensure the container is set to execute the PHP scripts, open your defined local host post in the browser,i.e., `http://localhost:8000/`.

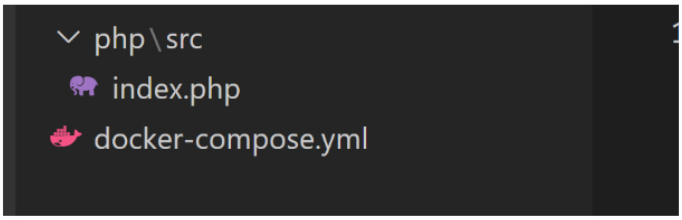
This shows the container is set to run some PHP-driven code.

Serve a dynamic PHP-driven website

Let's now execute some PHP code and get the output in the browser. You will be executing the scripts from the directory that you defined in the volumes of your docker-compose.

In this case we are using `./php/src`.

Head on to your project directory → `./php/src`, create an `index.php` file and start writing your PHP scripts.



A simple `index.php` script.

```php
<?php
echo "Hello there, this is a PHP Apache container";
?>
```

Refresh on your browser (`http://localhost:8000/`), and the results of this simple PHP drive website should be visible.

Congratulations! You now have a containerized PHP website.

Setup a MySQL database container

You would probably want to set up a database to interact with your

website. We will create another service to provide MySQL support inside the PHP container.

Let's add the MySQL service into the `docker-compose.yml` file. To setup MySQL, we need to customize some environment, such as:

- Password authentication. To use and access a MySQL server, you need to set authentication environments that will allow you to access the defined MySQL server and its services, such as a database. We will use `MYSQL_USER: MYSQL_USER` and `MYSQL_PASSWORD: MYSQL_PASSWORD` to connect to MySQL and access the `MYSQL_DATABASE: MYSQL_DATABASE`.
- A restart policy set to `restart: always`. This restarts the service whenever any defined configuration changes.

```
db:
    container_name: db
    image: mysql
    restart: always
    environment:
        MYSQL_ROOT_PASSWORD: MYSQL_ROOT_PASSWORD
        MYSQL_DATABASE: MY_DATABASE
        MYSQL_USER: MYSQL_USER
        MYSQL_PASSWORD: MYSQL_PASSWORD
    ports:
        - "9906:3306"
```

We need to add some MySQL support tools inside the PHP container for the two services (db and php-apache) to work correctly. This tool includes mysqli.

Inside your project directory, head to the `/php` folder, create a Docker file, name it `Dockerfile` and add the following PHP configurations.

```
FROM php:8.0-apache
RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
RUN apt-get update && apt-get upgrade -y
```

Here we have created a custom PHP Apache image and an environment that will install mysqli, a PHP extension that will connect the PHP Apache to the MySQL server.

Now we need to build this custom image inside php-apache service in the `docker-compose.yml` file. PHP Apache also depends on the `db` service to connect to MySQL. We need to configure it by specifying a `depends_on:` environment.

This is how your `docker-compose.yml` file should look like.

```
version: '3.8'
services:
    php-apache-environment:
        container_name: php-apache
        build:
            context: ./php
            dockerfile: Dockerfile
        depends_on:
            - db
        volumes:
            - ./php/src:/var/www/html/
        ports:
            - 8000:80
    db:
```

```
        container_name: db
        image: mysql
        restart: always
        environment:
            MYSQL_ROOT_PASSWORD: MYSQL_ROOT_PASSWORD
            MYSQL_DATABASE: MYSQL_DATABASE
            MYSQL_USER: MYSQL_USER
            MYSQL_PASSWORD: MYSQL_PASSWORD
        ports:
            - "9906:3306"
```

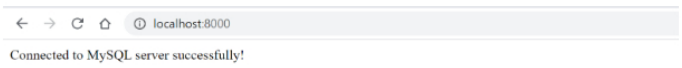
Run `docker-compose up` to pull and set up the MySQL environment. MySQL will be added to the container.

Run SQL query using PHP scripts

Let's test if the container is working as expected. Head over to the `index.php` file and the following PHP MySQL connection code.

```php
<?php
//These are the defined authentication environment in the db service

// The MySQL service named in the docker-compose.yml.
$host = 'db';

// Database use name
$user = 'MYSQL_USER';

//database user password
$pass = 'MYSQL_PASSWORD';

// check the MySQL connection status
$conn = new mysqli($host, $user, $pass);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
} else {
    echo "Connected to MySQL server successfully!";
}
?>
```

Save the file and refresh your `http://localhost:8000/` web address.



Boom. There you have it. The PHP Apache and MySQL environments are now set, and you can start developing your PHP-driven application and communicate with the MySQL server.

Setting PHPMyAdmin

The MySQL connection is now okay. Let's see how we can fetch some data

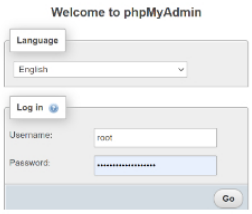
from a MySQL database and display it on a web page using PHP scripts.

Suppose your application interacts with a database; you would probably want an interface to interact with your data. We will add PHPMyAdmin services to provide us with an interface to interact with the MySQL database.

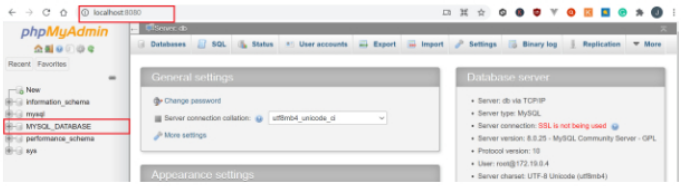
Let's add a PHPMyAdmin service as shown below.

```
phpmyadmin:
    image: phpmyadmin/phpmyadmin
    ports:
        - '8080:80'
    restart: always
    environment:
        PMA_HOST: db
    depends_on:
        - db
```

Open `http://localhost:8080/` on the browser to access the PHPMyAdmin.



To login to the Phpmyadmin panel, use username as `root` and password as `MYSQL_ROOT_PASSWORD`. The password was already set in the MySQL environment variables (`MYSQL_ROOT_PASSWORD: MYSQL_ROOT_PASSWORD`)



You can now see the database we defined is already set as `MYSQL_DATABASE`, and you can start interacting with Phpmyadmin.

Fetch records and print them on a PHP-driven website

Create a database table and fill in some records. Select the database and execute the following query.

```
drop table if exists `users`;
create table `users` (
    id int not null auto_increment,
    username text not null,
    password text not null,
    primary key (id)
);
insert into `users` (username, password) values
    ("admin","password"),
    ("Alice","this is my password"),
```

```
    ("Job","12345678");
```

Go ahead and write a select SQL query with PHP.

```php
<?php
//These are the defined authentication environment in the db service

// The MySQL service named in the docker-compose.yml.
$host = 'db';

// Database use name
$user = 'MYSQL_USER';

//database user password
$pass = 'MYSQL_PASSWORD';

// database name
$mydatabase = 'MYSQL_DATABASE';
// check the mysql connection status

$conn = new mysqli($host, $user, $pass, $mydatabase);

// select query
$sql = 'SELECT * FROM users';

if ($result = $conn->query($sql)) {
    while ($data = $result->fetch_object()) {
        $users[] = $data;
    }
}

foreach ($users as $user) {
    echo "<br>";
    echo $user->username . " " . $user->password;
    echo "<br>";
}
?>
```

Refresh `http://localhost:8000/` to view the results.

---

admin password

Alice this is my password

Job 12345678

We have only used the Read operation to demonstrate this. Go ahead and try other CRUD operations with the dockerized PHP application.

Conclusion

I hope this tutorial helped you set up a PHP and MySQL development environment using Docker containers.

Docker is awe-inspiring container technology with a tone of mesmerizing benefits. If you haven't started learning Docker yet, check out this tutorial and learn how to get started with Docker.

Feel free to check out the below tutorials and more about Docker and containers.

Happy coding!

- Understanding Docker Concepts
- Getting Started with Docker
- A Brief History of Container Technology

Peer Review Contributions by: Srishilesh P S

Did you find this article helpful?

146

26 Comments Sort By Best

The EngEd community is subject to Section's moderation policy.

Write your comment... LOGIN SIGNUP

ScottTrakker 1 year ago
Thank you for this excellent tutorial!

Two things I would like to add:

1. Use 'docker-compose up -d'. This way it starts detached from the terminal.
2. Make the database persistent by adding the code below. volumes:
    - dbdata:/var/lib/mysql

Instead of docker I used Podman (with podman-compose) and that worked fine!
Reply Share 1 0

Ricardo Lima 1 year ago
Very much helpful, Thanks
Reply Share 1 0

Fridtha 4 months ago
Joseph, you are officially THE MAN! This saved me DAYS of more work on something I had already been beating my head against for days before. Thank you. Thank you. Thank you.
Reply Share 0 0

khw 1 year ago
I usually don't register on a site to leave back, now I do (did)
Joseph, you saved my day
I just couldn't get the connection between composer and builder, if I may say so, which led to mysql not being part of the php installation.
Thx a big bunch
Reply Share 0 0

ROHIT TRIVEDI 1 year ago
Thanks for the help.
Reply Share 0 0

Charles 1 year ago
How to use another version of php, for example 7.4?
Reply Share 0 0

Chamath 1 year ago
Thanks a lot.
One suggestion to add. After changes in Dockerfile, which command need to run? I ran following command to rebuild docker
docker-compose up --build
Reply Share 0 0

David Magalhães 1 year ago
Thanks, it helped me a lot.
Reply Share 0 0

Joot 1 year ago

I'm facing this issue sigwinch shutting down gracefully docker. can you help me?

Reply Share 0 0

nieves 1 year ago

throws me this error
Warning: mysqli::__construct(): (HY000/1045): Access denied for user 'axsobd'@'172.18.0.4' (using password: YES) in /var/www/html/index_mysql.php on line 29
Connection failed: Access denied for user 'axsobd'@'172.18.0.4' (using password: YES)

My docker-compose is:
environment:
MYSQL_ROOT_PASSWORD: nieves
MYSQL_DATABASE: axsobd
MYSQL_USER: axsouser
MYSQL_PASSWORD: axso
ports:

- "3306:3306" /$link = mysqli_connect ("mysql:3306", $_ENV["axsouser"], $_ENV["axso"], $_ENV["axsobd"]) Help me, please

Reply Share 0 0

Chamath 1 year ago

I got same error. It seems, it hasn't run the Dockerfile. Try to build with docker-compose up --build

Reply Share 0 0

ScottTrakker 1 year ago

I try to run Laravel (PHP framework) with the above setup but when I run 'php artisan migrate' I get the error 'Class "PDO" not found'.

This is docker-compose file that I use:

version: 3.4
services:
php-apache-environment:
container_name: php-apache
build:
context: .
dockerfile: Dockerfile
depends_on:

- db volumes:
- ./www:/var/www/html:Z ports:
- 8000:80 db: container_name: db image: mysql restart: always environment: MYSQL_ROOT_PASSWORD: zqw36vuahnrk MYSQL_DATABASE: laravel-mvc MYSQL_USER: laravel-user MYSQL_PASSWORD: dw3kngbtd773 ports:
- 9906:3306 volumes:
- dbdata:/var/lib/mysql phpmyadmin: image: phpmyadmin/phpmyadmin ports:
- 8080:80 restart: always environment: PMA_HOST: db depends_on:
- db

With this Dockerfile:

FROM php:8.1-apache
RUN docker-php-ext-install pdo pdo_mysql
RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
RUN apt-get update && apt-get upgrade -y

I run the command 'php artisan migrate' on the host.
When I run 'php artisan migrate' inside the container I get the error
'Illuminate\Database\QueryException SQLSTATE[HY000] [2002] No such file or directory'.

Do somebody have an idea what I could do wrong / what I can do?

Reply Share 0 0

Art 1 year ago

try
RUN docker-php-ext-install pdo_mysql && docker-php-ext-enable pdo_mysql

Reply Share 0 0

Scott Trakker 1 year ago

Thanks, but I already fixed it!

This is now my Dockerfile:

```
FROM php:8.1-apache
RUN apt-get update
RUN apt-get install -y git libzip-dev zip unzip npm
RUN docker-php-ext-install pdo pdo_mysql zip
RUN a2enmod rewrite
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
```

And this is my docker-compose.yml file:

```
version: 3.4
```

```
services:
  webserver:
    container_name: webserver
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - database
    volumes:
      - ./www:/var/www/html:z
    ports:
      - 8000:80
  database:
    container_name: database
    image: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: [root password]
      MYSQL_DATABASE: [database name]
      MYSQL_USER: [database user]
      MYSQL_PASSWORD: [password user]
    ports:
      - 9906:3306
    volumes:
      - dbdata:/var/lib/mysql
  phpmyadmin:
    container_name: phpmyadmin
    image: phpmyadmin/phpmyadmin
    ports:
      - 8080:80
    restart: always
    environment:
      PMA HOST: database
```

Show More

Reply Share 0 0

r **roughi** 1 year ago

so good, thanks! you explained very nicely how to build up the docker-compose file, helped me a lot!

Reply Share 0 0

F **Felipe** 1 year ago

For those who are having issue trying to show de data from database, there is a little mistake on tutorial. On line **$mydatabase = 'MYSQL_DATABASE';** the correct is **$mydatabase = 'MY_DATABASE';**

Thanks Joseph, really helpful!!

Reply Share 0 0

s **sonali** 1 year ago

very helpful article. thanks a lot.

Reply Share 0 0

H **Harischandra Matara Kanka** 1 year ago

This is and Excellent Work. Very short, Simple, To the Point, Clearly Explained, and NO Errors. Great!

Reply Share 0 0

n **nima** 1 year ago

hi buddy
Your description is very useful and can do it without any error.
good job

Reply Share 0 0

M **Michael Møller** 1 year ago

Fantastic article. I have this error:
Fatal error: Uncaught Error: Class "mysqli" not found in /var/www/html/index.php:14 Stack trace: #0 {main} thrown in /var/www/html/index.php on line 14

Reply Share 0 0

ScottTrakker 1 year ago

I had the same error but when I waited a few seconds and reloaded the page, the problem was solved.

Reply Share 1 0

a **a. g f** 1 year ago

hi. good work. i Have the error:
Fatal error: Uncaught Error: Class "mysqli" not found in /var/www/html/index.php:14 Stack trace: #0 {main} thrown in /var/www/html/index.php on line 14

Reply Share 0 0

ScottTrakker 1 year ago

I had the same error but when I waited a few seconds and reloaded the page, the problem was solved.

Reply Share 1 0

C **Chamath** 1 year ago

You may need to build docker with this command.
docker-compose up --build

Reply Share 0 0

x **xx** 1 year ago

did you prepare your dockerfile??? read the article carefully:
FROM php:8.0-apache

RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
RUN apt-get update && apt-get upgrade -y

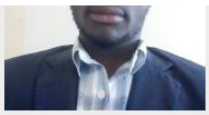
Reply Share 0 0

dudu 1 year ago

this dockerfile does not work...
docker-php-ext-install is not part of php:8.0-apache but php:8.0-alpine

Reply Share 0 0

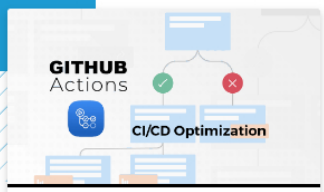
Similar Articles



Joseph Chege

Joseph Chege is an undergraduate student taking a Bachelor in Business Information Technology, a 4th-year student at Dedan Kimathi University of Technology. Joseph is fluent in Android Mobile Application Development and has a lot of passion for back-end development.

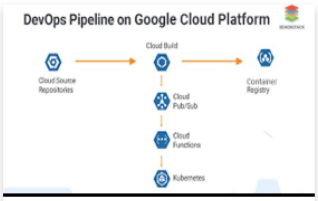
View author's full profile →



Containers

Continuous Integration and Deployment Pipelines with Flask, Docker and Github Actions

Read More →



Containers, API

DevOps Pipeline Automation with Google Cloud Build and Triggers

Read More →

Containers

Dockerizing an ASP.NET Core Web API App and SQL Server

Read More →