# **Dockerize Laravel Application**

## **#1 Dockerfile for Laravel**

You need to have your Laravel project in your local already setup, if you don’t, you can create one with this command.

composer create-project laravel/laravel --prefer-dist my\_app

Then, cd (change directory) to your project, and create a new file named Dockerfile. Notice that the name of the file starts with capital D without any extensions.

FROM php:7.4-fpm-alpine

RUN docker-php-ext-install pdo pdo\_mysql sockets

RUN curl -sS https://getcomposer.org/installer​ | php -- \

--install-dir=/usr/local/bin --filename=composer

COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

WORKDIR /app

COPY . .

RUN composer install

On line 1, we’ll use PHP version 7 as our base image. You can change the PHP version to whatever you want. Then on lines 3–7, that’s where you can install the dependencies that may be required in your project. Right now, we just need the composer for installing Laravel dependencies for our app. On lines 9–11, we change the directory into our app, then copy all the files from our local machine into the/app of our Docker container, then run composer install.

That’s it. We’ve created a custom image for our app. It’s time to run the image. You can run with or without *docker-compose*. For running our app without the *docker-compose*, here are the steps we can do.

First, open the terminal and type this to build your *Dockerfile*.

docker build -t my\_app .

Then, we may need to see the Image ID using this command.

docker image ls

Copy the image ID from the lists, and then we can run the image.

docker run -p 8000:8000 -d <imageID>

The flag -p above is exposing our main machine port, into the docker port. Then for the -d flag means that we’ll run the image in the background.

After the image is successfully running, we can run the server by executing the php artisan serve command inside the running container. First, we need to know what is the name (or container ID) of our app, by using this command docker ps . Then, we can get access to its command line using this command:

docker exec -it <container ID> sh

Inside the container command line, we can start the server using the usual php artisan serve.

php artisan serve --host=0.0.0.0

From your local machine, open localhost:8000 and you’ll see that our Laravel App is successfully running from the docker container.

## **#2 Running the Image with Docker Compose**

Create a new file named docker-compose.yaml inside the root project.

version: '3.8'

services:

main:

build:

context: .

dockerfile: Dockerfile

command: 'php artisan serve --host=0.0.0.0'

volumes:

- .:/app

ports:

- 8000:8000

For running the app, we can easily type this command:

docker-compose up

## **#3 MySQL DB & Artisan Queue Services**

Integrate our app into the database using the *docker-compose*. And if we need some additional commands to run other services, such as *artisan queue* to run the Laravel queue, we can do that also inside the *docker-compose*.

version: '3.8'

services:

main:

build:

context: .

dockerfile: Dockerfile

command: 'php artisan serve --host=0.0.0.0'

volumes:

- .:/app

ports:

- 8000:8000

depends\_on:

- db

queue:

build:

context: .

dockerfile: Dockerfile

command: 'php artisan queue:work'

depends\_on:

- db

db:

platform: linux/x86\_64

image: mysql:8.0

environment:

MYSQL\_DATABASE: main

MYSQL\_USER: admin

MYSQL\_ROOT: admin

MYSQL\_PASSWORD: admin

MYSQL\_ROOT\_PASSWORD: root

volumes:

- ./storage/dbdata:/var/lib/mysql

ports:

- 3306:3306