* [DEVOPS](https://javatodev.com/category/devops/)
* [SPRING BOOT](https://javatodev.com/category/springboot/)

**Docker Compose For Spring Boot with MySQL**

avatar

[**2**](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#comments)

In this article, I’ll explain how we can set up a deployment using docker-compose for a Spring Boot application which uses a **[MySQL](https://www.mysql.com/" \t "_blank)** database.

Technology stack which going to use in this tutorial,

* Spring Boot 2.4.1-RELEASE
* Spring Data JPA
* MySQL – 8.0
* Lombok
* Docker – version 19.03.14
* Docker-Compose – version 1.27.4
* IntellijIdea for IDE

Main topics inside this tutorial,

* [Setting Up MySQL Base Project](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#8ba655bb6a13)
* [Setup Docker Image For Spring Boot Application](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#f3b85a30e126)
* [Docker Compose File With Spring Boot and MySQL](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#da54d3217caf)
  + [Using links to Build Communication Between Services.](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#1b44a01d59a4)
  + [Using Depends On to Build Communication Between Services](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#3c62bc0a1a41)
* [Conclusion](https://javatodev.com/docker-compose-for-spring-boot-with-mysql/#8ccda91ea881)

Prerequisites,

You need to **setup** **docker** and **docker-compose** in your machine to go forward with this tutorial.

I’ve used following well written articles from **DigitalOcean** to setup both docker and docker-compose on my local setup.

* [How To Install and Use Docker on Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04)
* [How To Install Docker Compose on Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/how-to-install-docker-compose-on-ubuntu-18-04)

**Setting Up MySQL Base Project**

I’ve created a**simple REST API using Spring Boot** with MySQL database usage. First download or clone from [GitHub](https://github.com/javatodev/spring-boot-mysql-base-project" \t "_blank).

$ git clone https://github.com/javatodev/spring-boot-mysql-base-project.git

Or else you can skip this step and try a docker-compose set up **with your own spring boot application** which uses MySQL database.

**Setup Docker Image For Spring Boot Application**

In this tutorial, our docker-compose setup using two main services or components. Those are the **Spring Boot application**and**MySQL database.**

MySQL is already available as a docker image from [docker hub MySQL](https://hub.docker.com/_/mysql" \t "_blank) page. You can just go and select the [version](https://hub.docker.com/_/mysql?tab=tags&page=1&ordering=last_updated) you need from there.

But your **project is still not pushed or published as a docker image**. Hence first you need to have docker image for your spring boot application. Then you can use both app and DB images with docker-compose.

If you don’t have a good understanding of how to use docker with a spring boot application, you can refer our article on [**How to Dockerize Spring Boot Application**](https://javatodev.com/how-to-dockerize-spring-boot-application/).

OK Let’s start the docker image creation,

First, create a file naming as **Dockerfile** in your project root folder. Then copy the following content into that file.

**FROM** openjdk:8-jdk-alpine

**LABEL** maintainer="author@javatodev.com"

**VOLUME** /main-app

**ADD** build/libs/spring-boot-mysql-base-project-0.0.1-SNAPSHOT.jar app.jar

**EXPOSE** 8080

**ENTRYPOINT** ["java", "-jar","/app.jar"]

Now you have two ways of building docker compose setup.

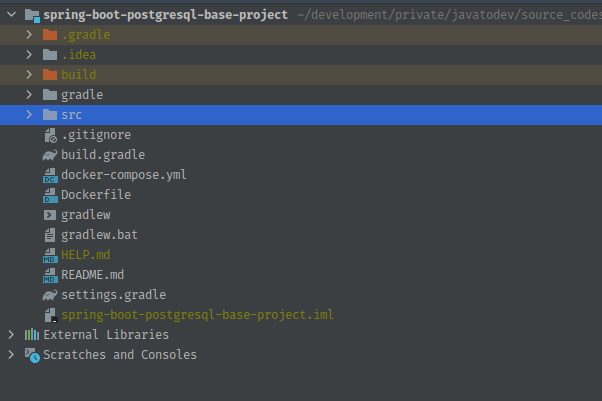
1. Pointing **Dockerfile** into the docker-compose setup.
2. Giving created docker image to the docker-compose setup.

**Let’s go forward with pointing docker file way, since giving docker image part will cover with MySQL image in this same setup.**

**Docker Compose File With Spring Boot and MySQL**

Now our application has the instruction to building the Docker image. So let’s create the **docker-compose.yml** file which **allows us to combine the Spring Boot application and MySQL** database in this setup.

First, go to the project root and create a file named **docker-compose.yml**

spring-boot-project-with-docker-compose-file

**Using links to Build Communication Between Services.**

There are **two ways of building communication** between spring boot application and the MySQL database**. You can use one from the following two methods to build this docker-compose setup**.

I’ll start with using links to build communication between both. So add the following content into the docker-compose.yml file,

**version:** "3.7"

**services:**

**api\_service:**

**build:** .

**restart:** always

**ports:**

- 8080:8080

**depends\_on:**

- mysql\_db

**links:**

- **mysql\_db:**app\_db

**mysql\_db:**

**image:** "mysql:8.0"

**restart:** always

**ports:**

- 3306:3306

**environment:**

**MYSQL\_DATABASE:** java\_to\_dev\_app\_db

**MYSQL\_USER:** java\_to\_dev

**MYSQL\_PASSWORD:** nE5kMc7JCGNqwDQM

**MYSQL\_ROOT\_PASSWORD:** nE5kMc7JCGNqwDQN

Here we have out two main services as,

1. **api\_service** – Spring Boot REST API which runs on port 8080.
2. **mysql\_db** – MySQL DB which runs on port 3306

Explanation about the configurations I have used in this docker-compose file,

**build** – Here we should introduce the image name to build under that service, Here we have our Dockerfile which developed for the API in the same level with docker-compose.yml so using “.” it will capture that docker file and build the handle the api\_service when executing this file.

Additionally, we have introduced **mysql:8.0** as the docker image for MySQL DB. It will download from docker registry and start when needed.

More to research: We can do the same to spring boot API docker image as well, Just push that image to docker-hub and you can directly use given image name and version for this setup as well.

**restart** – **always** Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted.

**ports** – Here we are mapping **local machine port** along with **port inside a docker container**.

**links** – Here we are setting a link with app\_db service. So we can use **database** when defining IP address for the database connection on our spring boot application. Then it will connect with the database running with this docker-compose.

**environment** – In this section, we are setting the MySQL database name, MySQL username and password for the application and root password.

All done, Now let’s change our **application.properties** to support this database and application server setup.

**spring.datasource.url=**jdbc:mysql://app\_db:3306/java\_to\_dev\_app\_db

**spring.datasource.username=**java\_to\_dev

**spring.datasource.password=**nE5kMc7JCGNqwDQM

**spring.jpa.hibernate.ddl-auto=**update

**spring.datasource.initialization-mode=**always

**Here as you can see we are using that link value we used in docker-compose definition for host/IP address**. additionally, password and DB has changed accordingly.

**Better Solution**: Use a **separate application.properties** file for docker-compose based deployments and set that **via profiles**.

**Using Depends On to Build Communication Between Services**

**The docker-compose documentation specifies that links is deprecated and should be replaced with depends\_on.**

So here I’m setting the same docker-compose setup with using depends\_on flag.

Just add following into the docker-compose.yml

**version:** "3.7"

**services:**

**api\_service:**

**build:** .

**restart:** always

**ports:**

- 8080:8080

**depends\_on:**

- mysql\_db

**command:** sh -c './wait-for mysql\_db:3306 -- npm start'

**mysql\_db:**

**image:** "mysql:8.0"

**restart:** always

**ports:**

- 3307:3306

**environment:**

**MYSQL\_DATABASE:** java\_to\_dev\_app\_db

**MYSQL\_USER:** java\_to\_dev

**MYSQL\_PASSWORD:** nE5kMc7JCGNqwDQM

**MYSQL\_ROOT\_PASSWORD:** nE5kMc7JCGNqwDQN

Here we are using depends\_on to set dependent services to api\_service. So internally docker-compose will start dependency services first and it will start dependent service in the end. So basically here it will start the DB first and finally application will be started.

Additionally we have a small change on database connection properties, Here you should add service name for the host or IP on that database connection.

**spring.datasource.url=**jdbc:mysql://mysql\_db:3306/java\_to\_dev\_app\_db

**spring.datasource.username=**java\_to\_dev

**spring.datasource.password=**nE5kMc7JCGNqwDQM

**spring.jpa.hibernate.ddl-auto=**update

**spring.datasource.initialization-mode=**always

**Running Spring Boot Application and MySQL Database Using Docker Compose**

Now we have our docker compose setup for this application. So first create a jar build for this application using the following command,

Navigate to application root folder and execute,

$ ./gradlew clean build

Now there should be a newly created jar file with all the necessary files to run this application on **build/libs** folder.

Then create the build with docker compose to build docker image using built jar file.

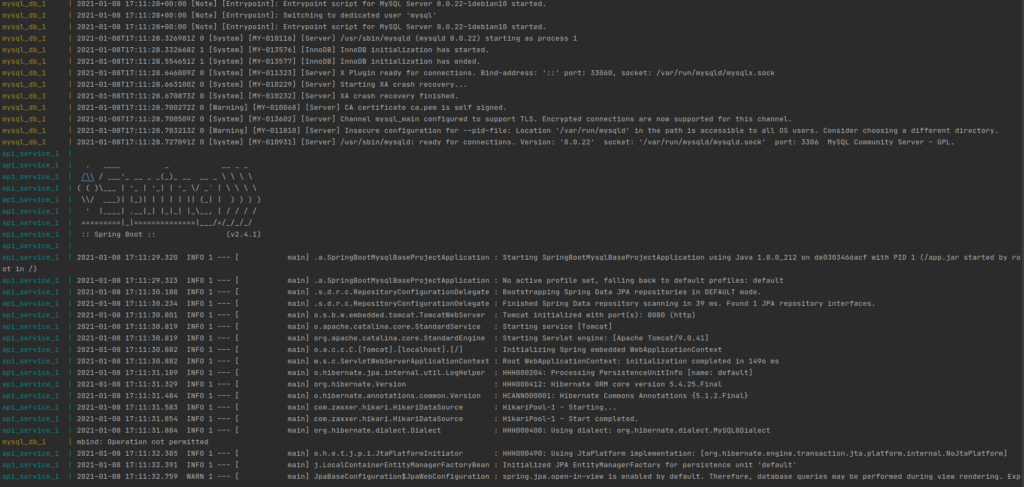
$ docker-compose build

Then use following command to run whole setup using docker compose.

$ docker-compose up

Then It will capture the docker-compose.yml and start running using the instructions given on that file.

sample output:

docker compose up spring boot mysql output