Docker Compose: Spring Boot and MySQL example

bezkoder.com/docker-compose-spring-boot-mysql

bezkoder Last modified: September 2, 2021

Docker provides lightweight containers to run services in isolation from our infrastructure so we can deliver software quickly. In this tutorial, I will show you how to dockerize Spring Boot microservice and MySQL example using Docker Compose.

Related Posts:
– Spring Boot, Spring Data JPA, MySQL – Rest CRUD API example
– Spring Boot Token based Authentication with Spring Security & JWT
– Spring Boot + GraphQL + MySQL example
– Spring Boot Rest XML example – Web service with XML Response
– Spring Boot: Upload CSV file data into MySQL Database
– Spring Boot: Upload Excel file data into MySQL Database

AWS instead: Deploy Spring Boot App on AWS – Elastic Beanstalk

Overview

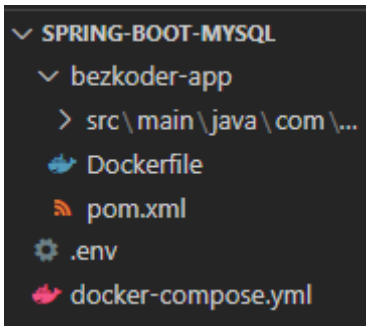
Assume that we have a Spring Boot Application working with MySQL database.
The problem is to containerize a system that requires more than one Docker container:

- Spring Boot for Rest API
- MySQL for database

Docker Compose helps us setup the system more easily and efficiently than with only Docker. We're gonna following these steps:

- Create Spring Boot App working with MySQL database.
- Create Dockerfile for Spring Boot App.
- Write Docker Compose configurations in YAML file.
- Set Spring Boot Docker Compose Environment variables.
- Run the system.

Directory Structure:



Create Spring Boot App

You can read and get Github source code from one of following tutorials:
– Spring Boot, Spring Data JPA, MySQL – Rest CRUD API example
– Spring Boot Token based Authentication with Spring Security & JWT
– Spring Boot + GraphQL + MySQL example
– Spring Boot Rest XML example – Web service with XML Response
– Spring Boot: Upload CSV file data into MySQL Database
– Spring Boot: Upload Excel file data into MySQL Database

Using the code base above, we put the Spring Boot project in **bezkoder-app** folder without the need of **resources**/*application.properties*. It is because Environment variables will be exported to *.env* file.

Create Dockerfile for Spring Boot App

Dockerfile defines a list of commands that Docker uses for setting up the Spring Boot application environment. So we put the file in **bezkoder-app** folder.

Because we will use Docker Compose, we won't define all the configuration commands in this Dockerfile.

bezkoder-app/*Dockerfile*

```
FROM maven:3.8.2-jdk-8
WORKDIR /bezkoder-app
COPY . .
RUN mvn clean install
CMD mvn spring-boot:run
```

Let me explain some points:

- `FROM` : install the image of the Maven – JDK version.
- `WORKDIR` : path of the working directory.
- `COPY` : copy all the files inside the project directory to the container.
- `RUN` : execute a command-line inside the container: `mvn clean install` to install the dependencies in *pom.xml*.
- `CMD` : run script `mvn spring-boot:run` after the image is built.

Write Docker Compose configurations

On the root of the project directory, we're gonna create the *docker-compose.yml* file. Follow version 3 syntax defined by Docker:

```
version: '3.8'
services:
    mysqldb:
    app:
volumes:
```

- `version` : Docker Compose file format version will be used.
- `services` : individual services in isolated containers. Our application has two services: `app` (Spring Boot) and `mysqldb` (MySQL database).
- `volumes` : named volumes that keeps our data alive after restart.

Let's implement the details.

docker-compose.yml

```
version: "3.8"
services:
  mysqldb:
    image: mysql:5.7
    restart: unless-stopped
    env_file: ./.env
    environment:
      - MYSQL_ROOT_PASSWORD=$MYSQLDB_ROOT_PASSWORD
      - MYSQL_DATABASE=$MYSQLDB_DATABASE
    ports:
      - $MYSQLDB_LOCAL_PORT:$MYSQLDB_DOCKER_PORT
    volumes:
      - db:/var/lib/mysql
  app:
    depends_on:
      - mysqldb
    build: ./bezkoder-app
    restart: on-failure
    env_file: ./.env
    ports:
      - $SPRING_LOCAL_PORT:$SPRING_DOCKER_PORT
    environment:
      SPRING_APPLICATION_JSON: '{
        "spring.datasource.url"  :
"jdbc:mysql://mysqldb:$MYSQLDB_DOCKER_PORT/$MYSQLDB_DATABASE?useSSL=false",
        "spring.datasource.username" : "$MYSQLDB_USER",
        "spring.datasource.password" : "$MYSQLDB_ROOT_PASSWORD",
        "spring.jpa.properties.hibernate.dialect" :
"org.hibernate.dialect.MySQL5InnoDBDialect",
        "spring.jpa.hibernate.ddl-auto" : "update"
      }'
    volumes:
      - .m2:/root/.m2
    stdin_open: true
    tty: true
volumes:
  db:
```

– **mysqldb**:

- `image` : official Docker image
- `restart` : configure the restart policy
- `env_file` : specify our *.env* path that we will create later
- `environment` : provide setting using environment variables
- `ports` : specify ports will be used
- `volumes` : map volume folders

– **app**:

- `depends_on` : dependency order, **mysqldb** is started before **app**
- `build` : configuration options that are applied at build time that we defined in the *Dockerfile* with relative path
- `environment` : environmental variables that Spring Boot application uses
- `stdin_open` and `tty` : keep open the terminal after building container

You should note that the host port (`LOCAL_PORT`) and the container port (`DOCKER_PORT`) is different. Networked service-to-service communication uses the container port, and the outside uses the host port.

Docker Compose Environment variables

In the service configuration, we used environmental variables defined inside the *.env* file. Now we start writing it.

.env

```
MYSQLDB_USER=root
MYSQLDB_ROOT_PASSWORD=123456
MYSQLDB_DATABASE=bezkoder_db
MYSQLDB_LOCAL_PORT=3307
MYSQLDB_DOCKER_PORT=3306
SPRING_LOCAL_PORT=6868
SPRING_DOCKER_PORT=8080
```

Run the Spring Boot microservice with Docker Compose

We can easily run the whole with only a single command:
`docker-compose up`

Docker will pull the MySQL and Maven images (if our machine does not have it before).

The services can be run on the background with command:
`docker-compose up -d`

```
$ docker-compose up -d
Creating network "spring-boot-mysql_default" with the default driver
Creating volume "spring-boot-mysql_db" with default driver
Pulling mysqldb (mysql:5.7)...
5.7: Pulling from library/mysql
e1acddbe380c: Pull complete
bed879327370: Pull complete
03285f80bafd: Pull complete
ccc17412a00a: Pull complete
1f556ecc09d1: Pull complete
adc5528e468d: Pull complete
1afc286d5d53: Pull complete
4d2d9261e3ad: Pull complete
ac609d7b31f8: Pull complete
53ee1339bc3a: Pull complete
b0c0a831a707: Pull complete
Digest: sha256:7cf2e7d7ff876f93c8601406a5aa17484e6623875e64e7acc71432ad8e0a3d7e
Status: Downloaded newer image for mysql:5.7
Building app
Sending build context to Docker daemon  22.02kB
Step 1/5 : FROM maven:3.8.2-jdk-8
 ---> 80704b8c5fbd
Step 2/5 : WORKDIR /bezkoder-app
 ---> Running in f63e76f45fcc
Removing intermediate container f63e76f45fcc
 ---> 10802ac64cea
Step 3/5 : COPY . .
 ---> 9dcd16082f00
Step 4/5 : RUN mvn clean install
 ---> Running in 288bea890f74
[INFO] Scanning for projects...
Downloading from central:
https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-
parent/2.2.1.RELEASE/spring-boot-starter-parent-2.2.1.RELEASE.pom
Downloaded from central:
https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-
parent/2.2.1.RELEASE/spring-boot-starter-parent-2.2.1.RELEASE.pom (8.1 kB at 4.2
kB/s)
Downloading from central:
https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-
dependencies/2.2.1.RELEASE/spring-boot-dependencies-2.2.1.RELEASE.pom
Downloaded from central:
https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-
dependencies/2.2.1.RELEASE/spring-boot-dependencies-2.2.1.RELEASE.pom (127 kB at
201 kB/s)
...
[INFO] Installing /bezkoder-app/target/spring-boot-data-jpa-0.0.1-SNAPSHOT.jar to
/root/.m2/repository/com/bezkoder/spring-boot-data-jpa/0.0.1-SNAPSHOT/spring-boot-
data-jpa-0.0.1-SNAPSHOT.jar
[INFO] Installing /bezkoder-app/pom.xml to
/root/.m2/repository/com/bezkoder/spring-boot-data-jpa/0.0.1-SNAPSHOT/spring-boot-
data-jpa-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:41 min
[INFO] Finished at: 2021-08-18T04:10:08Z
[INFO] ------------------------------------------------------------------------
Removing intermediate container 288bea890f74
```

```
 ---> adddf4648410
Step 5/5 : CMD mvn spring-boot:run
 ---> Running in c81f8028e2eb
Removing intermediate container c81f8028e2eb
 ---> 1f710daedbf2
Successfully built 1f710daedbf2
Successfully tagged spring-boot-mysql_app:latest
WARNING: Image for service app was built because it did not already exist. To
rebuild this image you must use `docker-compose build` or `docker-compose up --
build`.
Creating spring-boot-mysql_mysqldb_1 ... done
Creating spring-boot-mysql_app_1     ... done
```

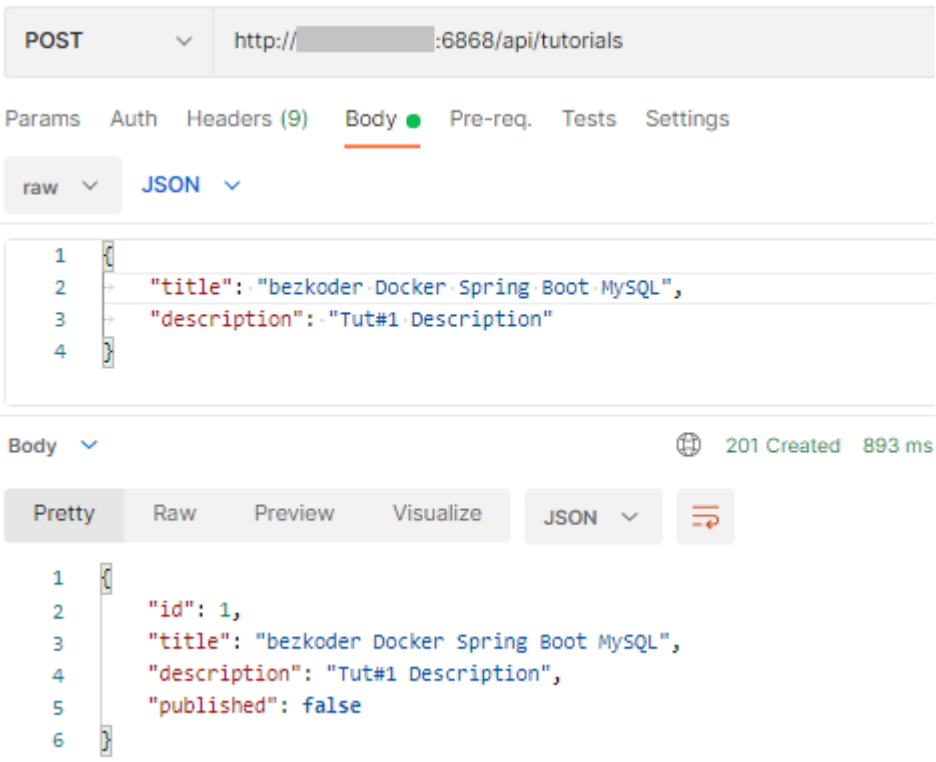
Now you can check the current working containers:

```
$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED
STATUS         PORTS                                                NAMES
5ad28f104e8b   spring-boot-mysql_app   "/usr/local/bin/mvn-…"   3 minutes ago   Up
3 minutes   0.0.0.0:6868->8080/tcp, :::6868->8080/tcp             spring-boot-
mysql_app_1
ba9281773e7f   mysql:5.7              "docker-entrypoint.s…"   3 minutes ago   Up
3 minutes   33060/tcp, 0.0.0.0:3307->3306/tcp, :::3307->3306/tcp   spring-boot-
mysql_mysqldb_1
```

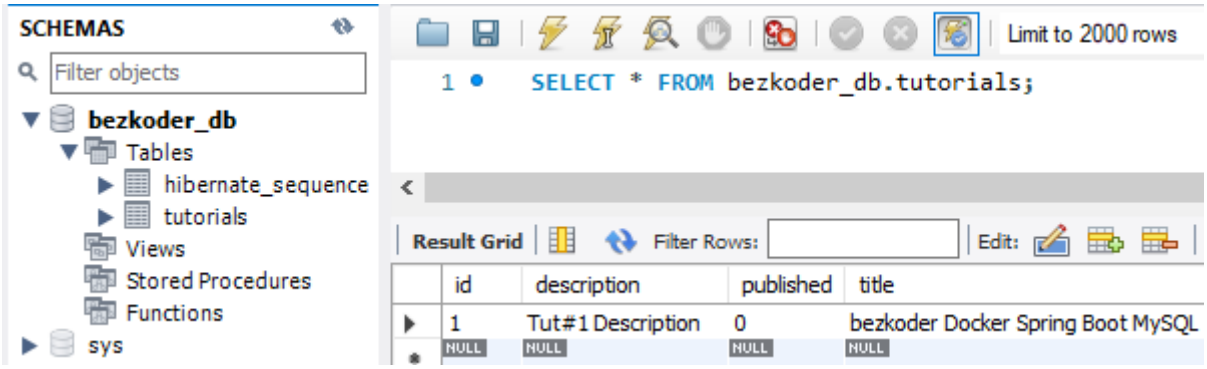
And Docker images:

```
$ docker images
REPOSITORY              TAG            IMAGE ID       CREATED         SIZE
spring-boot-mysql_app   latest         1f710daedbf2   5 minutes ago   672MB
mysql                   5.7            6c20ffa54f86   6 minutes ago   448MB
maven                   3.8.2-jdk-8    80704b8c5fbd   6 minutes ago   525MB
```

Send a HTTP request to the Spring Boot – MySQL system:



Check MySQL Database:



Stop the Application

Stopping all the running containers is also simple with a single command:
`docker-compose down`

```
$ docker-compose down
Stopping spring-boot-mysql_app_1     ... done
Stopping spring-boot-mysql_mysqldb_1 ... done
Removing spring-boot-mysql_app_1     ... done
Removing spring-boot-mysql_mysqldb_1 ... done
Removing network spring-boot-mysql_default
```

If you need to stop and remove all containers, networks, and all images used by any service in *docker-compose.yml* file, use the command:
`docker-compose down --rmi all`

Conclusion

Today we've successfully created Docker Compose file for Spring Boot application and MySQL. Now we can connect Spring Boot to MySQL with Docker on a very simple way: *docker-compose.yml*.

You can apply this way to one of following project:
– Spring Boot, Spring Data JPA, MySQL – Rest CRUD API example
– Spring Boot Token based Authentication with Spring Security & JWT
– Spring Boot + GraphQL + MySQL example
– Spring Boot Rest XML example – Web service with XML Response
– Spring Boot: Upload CSV file data into MySQL Database
– Spring Boot: Upload Excel file data into MySQL Database

If you want to deploy the system on AWS, please visit:
Deploy Spring Boot App on AWS – Elastic Beanstalk

Happy Learning! See you again.

Source Code

The source code for this tutorial can be found at Github.

You can deploy the container on Digital Ocean with very small budget: **5$**/month. Using referral link below, you will have **100$** in credit over **60** days. After that, you can stop the VPS with no cost.