Using Docker Compose with Spring Boot and PostgreSQL

Gozde Saygili Yalcin

.

Follow

5 min read

.

Dec 5, 2023

--

Listen

Share

In today's software world, Docker Compose makes it easy to handle Spring Boot apps in Docker containers, especially complex ones. You **can describe multiple containers as one application and start everything with a single command**.

In this article, the fundamental usage of Docker Compose for creating a standard development environment will be examined;

- **What is Docker Compose?:** A brief explanation.
- **Why Does It Exist?:** Understanding its purpose and how it's different from "docker run".
- **How to Use It:** A quick guide on using Docker Compose.
- **When to Use It:** Exploring practical use cases.

What is Docker Compose?

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

It's like creating **a blueprint** for how your containers should work together, and then **using a simple command to start them** all at once.

Before diving deep, let's try to understand key concepts about docker.

Docker Terms

Dockerfile: This is a text file that contains **instructions on how to build a docker image**.

Docker image: This is a **snapshot of the computer program and everything** it needs to run embedded in it.

Docker Container: This is a **lightweight, stand-alone executable package** that includes everything needed to run a piece of software, such as code, runtime, system tools, libraries, and settings.

For further terms, can be checked it here:

https://docs.docker.com/glossary/

Why Does It Exist

Some of the notable features of Compose that enhance its effectiveness include:

- **Isolating Multiple Environments:** Compose allows you to set up and manage multiple isolated environments on a single host.
- **Data Volume Preservation:** It retains volume data when containers are created or modified, ensuring data persistence.
- **Efficient Container Recreations:** Compose doesn't redo everything, just the parts that have changed.
- **Environment Flexibility:** It can work with different settings and can be moved to different places easily.

How to Use It

Step 1: Install Docker & Docker Compose

- Docker

- Docker Compose

Step 2: Create a sample web application

Starting a Spring Boot 3 project has a few setup options. The starter you can find in the link below is preferred.

- Spring initializer

In order to use PostgreSQL, the following dependency has been added.

```
        <dependency>
           <groupId>org.postgresql</groupId>
           <artifactId>postgresql</artifactId>
           <scope>runtime</scope>
       </dependency>
```

To use REST endpoints for our application, the web dependency has been added.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Step 3: Create docker-compose.yml and Dockerfile

In this example, a simple customer project has been created to test the project. At the end, the project package is divided as follows;

```
my-docker-project/
|-- src/
|   |-- main/
|       |-- java/
|           |-- com/
|               |-- gozde/
|                   |-- dockercomposedemo/
|                       |-- controller/
|                           |-- CustomerController.java
|                       |-- model/
|                           |-- Customer.java
|                       |-- model/
|                           |-- dto/
|                               |-- CreateCustomerDto.java
|                       |-- service/
|                           |-- CustomerService.java
|-- docker-compose.yml
|-- Dockerfile
```

docker-compose.yaml

The **docker-compose.yml** file is a YAML file that specifies services, networks, and volumes for a Docker container. There are multiple versions of the compose file format, including versions 1, 2, 2.x, and 3.x.

In this sample, we have got '**db' (postgres)** and **'app'** services. 'db' uses the official PostgreSQL image with a volume for data. 'app' builds our Spring Boot app using the Dockerfile, with defined environment variables for the database connection

```
version: '3.8'

services:
 app:
   image: 'docker-spring-boot-postgres:latest'
   build:
     context: .
   container_name: app
   depends_on:
     - db
   environment:
     - POSTGRES_USER=$YOUR_USERNAME
     - POSTGRES_PASSWORD=$YOUR_PASSWORD
     - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/$YOUR_DB_NAME
     - SPRING_DATASOURCE_USERNAME=$YOUR_USERNAME
     - SPRING_DATASOURCE_PASSWORD=$YOUR_PASSWORD
     - SPRING_JPA_HIBERNATE_DDL_AUTO=update
```

```
    ports:
      - "$LOCAL_PORT:$DOCKER_PORT"

  db:
    image: postgres
    container_name: db
    environment:
      - POSTGRES_USER=$YOUR_USERNAME
      - POSTGRES_PASSWORD=$YOUR_PASSWORD
      - POSTGRES_DB=$YOUR_DB_NAME
    ports:
      - "$LOCAL_PORT:$DOCKER_PORT"
```

Let's understand the yaml step by step;

- `version` refers to the docker-compose version (Latest 3)
- `services` defines the services that we need to run
- `app` is a custom name for one of the containers
- `image` the image which we have to pull. Here we are using `postgres` .
- `container_name` is the name for each container
- `port` defines the custom port to run the container
- `environment` defines the environment variables, such as DB credentials, and so on.

Dockerfile

To build an image for our Spring Boot application, creating a new file named **Dockerfile** in the project's root directory.

```
# the base image
FROM amazoncorretto:17

# the JAR file path
ARG JAR_FILE=target/*.jar

# Copy the JAR file from the build context into the Docker image
COPY ${JAR_FILE} application.jar

CMD apt-get update -y

# Set the default command to run the Java application
ENTRYPOINT ["java", "-Xmx2048M", "-jar", "/application.jar"]
```

Step 4: Build and Run the Application

Now that we have set up the **Dockerfile** and **Docker Compose** file, we are ready to build and run the application with Docker Compose. Open your terminal and run the following command in the root directory of the project:

```
$ docker-compose up
```

`docker-compose up --build`: Starts your services and, at the same time, rebuilds Docker images based on your Dockerfiles. It ensures you're using the latest changes in your code or dependencies.

As you can see, the application is built using Docker Compose.

Once the containers are running, you should be able to access the application at http://localhost:6868/customers

`docker ps` command is used to check running containers.`docker-compose ps` provides a summary of the status of services defined in a **Docker Compose** file, can see the output of these commands;

In the list of containers, you should also be able to see both the database and application containers.

List of containers

Lastly, the `docker-compose down` command stops and removes the containers, networks, and volumes created by your Docker Compose project. It cleans up the resources, freeing up system space and stopping your services. When running docker-compose down, you will notice that the containers' names are no longer listed.

Conclusion

In this article, the steps for executing a Spring Boot application and PostgreSQL database in Docker using Docker Compose are implemented, which simplifies the definition and management of multi-container applications.

Source code for this demo: https://github.com/GozdeSaygiliYalcin/docker-compose-demo

You can use the following references for further reading.

References:

https://docs.docker.com/compose/

https://docs.docker.com/get-started/08_using_compose/#:~:text=Docker%20Compose%20is%20a%20tool,or%20tear%20it%20all%20down.

https://www.baeldung.com/ops/docker-compose-support-spring-boot

https://techmormo.com/posts/what-is-docker-compose/

https://phoenixnap.com/kb/docker-compose

https://www.freecodecamp.org/news/what-is-docker-compose-how-to-use-it/

https://springhow.com/spring-boot-and-postgres-using-docker-compose/#google_vignette

https://medium.com/aws-in-plain-english/docker-explained-simply-for-a-10-year-old-the-magic-box-for-computer-programs-94452b930d6b