Spring Boot 3.1 — Integrated Docker Compose on Development environment (Spring WebFlux example)

Thanaphoom Babparn

·

Follow

6 min read
·
Apr 27, 2023

--

2

Listen

Share

Hello everyone, this article is all about Spring Boot. The Example Project that I have created for this article is built using Spring WebFlux also. Recently, Spring Team has recently updated the version of Spring Boot to 3.1.0 and has so many things quite interesting.

Spring Boot 3.1.0 RC1 Release Notes

You can’t perform that action at this time. You signed in with another tab or window. You signed out in another tab or…

github.com

I would like to try creating a small short-term project that everyone can update along with me. ■

I will discuss two things here:

1 — Testcontainers: The tool that we use for Spring Boot Integration Tests. It allows us to use **spring-boot-testcontainers** in development time instead of just using it for testing.

More information about spring-boot-testcontainers

2 — spring-boot-docker-compose: This is a new library that supports Docker Compose files in the project. When the application starts, it searches for the Docker Compose file and configuration and creates Testcontainers for us. Additionally, it attaches these configurations to our local application. This means that we can run the docker-compose.yaml in our project without having to configure it in the application.yaml.

More information about spring-boot-docker-compose

This format is essentially a way of **omitting configuration properties and then using environment properties during high-level deployment**.

I don’t want to wasting everyone time, I have prepared a walkthrough of the project I created for this purpose.

Example Project

GitHub — marttp/20230427-spring-webflux-docker-compose: Example project to used docker compose…

You can’t perform that action at this time. You signed in with another tab or window. You signed out in another tab or…

github.com

The project will consist of the following:

- Spring WebFlux
- Spring Data R2DBC with PostgreSQL
- Spring Data Redis Reactive
- JDK 17

The concept is to implement **the Book Service with the Cache-Aside pattern**. The main actor of this repository is the following.

```
implementation("org.springframework.boot:spring-boot-docker-compose")
```

Above module, will allow our Spring Boot application to **scan all the information from the Docker Compose file without having to configure it in applications.yaml (or application.properties).** However, this is only applicable to the development environment. It’s important to note that if we deploy at a higher level, we need to configure it with the write properties to avoid issues.

Part 1 — Deployment Config -> Able to Run application

Below is my docker-compose.yaml

Next, we need to set up the schema configuration. We can create a schema.sql file in the src/main/resources folder to define the schema for our database.

application.yaml

```
spring:
 sql:
   init:
     schema-locations: classpath:schema.sql
     mode: always
```

schema.sql

```
CREATE TABLE IF NOT EXISTS book (
  id SERIAL PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  isbn VARCHAR(20) NOT NULL
);
```

Start application…ehhh…What?

Ahhhh…I see. Although we have included the necessary libraries, our machine still needs to have a container host running. Therefore, the prerequisite for using these libraries is to install Docker on our machine. This is necessary for Testcontainers to run properly as well.

During the startup of the application, the console will show that images are being pulled if they do not exist on our machine.

Application started successfully. No errors. ■

Checking the containers process with following command

```
docker ps
```

Yes, we should see that a container has been created for us ■. This means that Testcontainers has successfully started a container for our application to use.

If we stop the application, the container will also stop and be removed. This is because Testcontainers is managing the container’s lifecycle and ensures that it is cleaned up properly when it is no longer needed.

Part 2 — Business Logic work as expected?

As mentioned earlier, to implement the Cache-Aside pattern, we need a Cache and a database.

Additional information: the Cache-Aside pattern is a way of accessing data stored in a cache. When the data is not found in the cache, it is retrieved from the database and stored in the cache for faster access in subsequent requests.

The main implementation codes for the Cache-Aside pattern will typically include:

Next, an **ApplicationRunner** bean is created to initialize some test data. **Note that in a production environment, the deleteAll method should not be used.** This is only for testing purposes.

The result of the implementation should look similar to the screenshot below:

Now let’s test if the Cache-Aside pattern implementation is working correctly using HTTP requests. You can use Postman or cURL as per your preference to send HTTP requests.

I will use RedisInsight to access and view the cached data in Redis.

RedisInsight

Visualize and optimize Redis data RedisInsight is a powerful tool for visualizing and optimizing data in Redis or Redis…

redis.io

For those who are not familiar with RedisInsight, it is a GUI tool provided by Redis team for accessing Redis databases. It provides various functions for querying and mutating data in Redis.

To use RedisInsight, you can download and install it from the official website. After installation, you can launch the application and connect to the Redis instance that your Spring Boot application is using for caching.

Using RedisInsight can be very helpful for developers who are working with Redis databases, as it provides a user-friendly interface for managing and manipulating Redis data.

Here the last step, After sending the request, you can see that the data has been stored in the cache as expected ■. This means that the Cache-Aside pattern has been successfully implemented and the data is being retrieved from the cache when it is available, and from the database when it is not.

Summary

In this article, I shared my experience working on a Book Service project using the Cache-Aside pattern with Spring Boot, Spring WebFlux, Spring Data R2DBC, Spring Data Redis Reactive, and JDK 17.

As from the example I gave, I haven’t applied GraalVM yet. However, everyone can clone and try it out for themselves, including using the Kafka image for testing. **The only thing I’m curious about in this POC is whether it’s really useful or not**.

Also, apart from this, I may have written some parts poorly and there are some areas that need improvement. If everyone want to make any adjustments by yourself, feel free to try and adjust it as by your fit. ✌■

I think that’s about it for this article. See you again in the next articles. Thank you to everyone who read until the end. I wish everyone happiness ■

Best regards,

Thanaphoom Babparn

Facebook: Thanaphoom Babparn
FB Page: TP Coder
LinkedIn: Thanaphoom Babparn
Linktree: https://linktr.ee/tpbabparn