

Dockerizing Your Maven Spring Boot Application: A Step-by-Step Guide



Gürsel Gazi

Follow

3 min read

Mar 12, 2024

Listen

Share

Containerization has revolutionized the way we develop, deploy, and scale applications. Docker, a leading platform in this arena, encapsulates applications along with their environments, ensuring consistency across different deployment scenarios. For Java developers, integrating Docker with a Maven-based Spring Boot application can streamline workflows and enhance deployment strategies. This article provides a comprehensive guide to dockerizing your Maven Spring Boot application.

Understanding the Basics

Before we dive into the dockerization process, let's establish a basic understanding of the key components:

- **Spring Boot:** A framework for building standalone, production-grade Spring-based Applications with minimal configuration.
- **Maven:** A build automation tool used primarily for Java projects.
- **Docker:** A platform for developing, shipping, and running applications inside containers.

Prerequisites

- Java Development Kit (JDK) installed
- Spring Boot application built with Maven
- Docker installed on your machine

Step 1: Prepare Your Spring Boot Application

Ensure your Spring Boot application is functioning correctly and can be built using Maven. Your `pom.xml` should define the project and its dependencies. Here's a simple example:

```
<project>
  <!-- Project details -->
  <groupId>com.example</groupId>
  <artifactId>spring-boot-app</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
  </parent>

  <!-- Dependencies -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Other dependencies -->
  </dependencies>
```

```
</project>
```

Step 2: Create a Dockerfile

The Dockerfile is a script containing a series of instructions on how to build the Docker image for your application. Create a file named `Dockerfile` in the root directory of your Spring Boot project and add the following content:

```
# Start with a base image containing Java runtime
FROM openjdk:8-jdk-alpine

# Add Maintainer Info
LABEL maintainer="your.email@example.com"

# Add a volume pointing to /tmp
VOLUME /tmp

# Make port 8080 available to the world outside this container
EXPOSE 8080

# The application's jar file
ARG JAR_FILE=target/spring-boot-app-1.0.0.jar

# Add the application's jar to the container
ADD ${JAR_FILE} app.jar

# Run the jar file
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]
```

This Dockerfile uses the OpenJDK 8 image as a base, exposes port 8080, copies your application's JAR file into the image, and specifies how to run your application.

Step 3: Build the Docker Image

With the Dockerfile in place, navigate to the root directory of your Spring Boot application in the terminal and run the following command to build the Docker image:

```
docker build -t spring-boot-app .
```

This command builds a new Docker image named `spring-boot-app` based on the instructions in your Dockerfile.

Step 4: Run Your Docker Container

Once the image is built, you can run your Spring Boot application inside a Docker container using:

```
docker run -p 8080:8080 spring-boot-app
```

This command starts a new container from the `spring-boot-app` image, mapping port 8080 of the container to port 8080 on your host, allowing you to access the application as you would normally.

Step 5: Verify the Application

Ensure that your Spring Boot application is running correctly inside the container. Open your web browser and navigate to `http://localhost:8080` to view your application's homepage or whatever endpoint you have defined.

Conclusion

Dockerizing your Maven Spring Boot application can significantly simplify the development and deployment process, making your applications more portable and scalable. By following the steps outlined in this guide, you can containerize your Spring Boot application, ensuring it runs consistently across different environments. Happy containerizing!