Docker

Table of Contents

Introduction
Docker is a powerful tool that allows you to package applications, including Java Spring Boot projects, into containers. These containers ensure that your application runs consistently across various environments, making Docker an essential tool for DevOps and cloud deployments.

Core DevOps Principles and Practices
Docker fits into the broader DevOps framework, which includes:
- Continuous Integration (CI): Integrating code changes frequently to a shared repository.
- Continuous Delivery (CD): Automating the release process to deploy applications quickly.
- Infrastructure as Code (IaC): Defining and managing infrastructure through code (like Dockerfiles).
- Monitoring: Keeping track of applications' performance in production.
- Collaboration: Facilitating communication between developers and IT operations.

Building and Running Docker Images for Spring Boot Applications
To containerize a Spring Boot application, follow these steps:

1. **Create a Dockerfile**
   A `Dockerfile` is used to create Docker images. For a Spring Boot project, the typical `Dockerfile` might look like this:

   ```dockerfile
   FROM openjdk:17-jdk-slim
   ARG JAR_FILE=target/*.jar
   COPY ${JAR_FILE} app.jar
   ENTRYPOINT ["java","-jar","/app.jar"]
   ```
   This Dockerfile:
   - Uses the official OpenJDK 17 image.
   - Copies the compiled Spring Boot JAR file into the container.
   - pecifies the command to run the JAR file when the container starts.
2. **Build the Docker Image**
   Use the `docker build` command to build your image:
   ```bash
   docker build -t my-spring-boot-app .
   ```
   This command builds the image and tags it as `my-spring-boot-app`.

3. Run the Docker Container
   After building the image, run it using:

   ```
   docker run -d -p 8080:8080 --name springboot-app my-spring-boot-app
   ```

   This command:
   - Runs the container in detached mode (`-d`).
   - Maps port 8080 on the host to port 8080 in the container.
   - Names the container `springboot-app`.

Multi-Container Spring Boot Application with Docker Compose
For more complex applications that involve multiple services (e.g., a Spring Boot app with a PostgreSQL database), you can use Docker Compose. Here's a typical `docker-compose.yml` file for a Spring Boot application:

```yaml
version: '3'
services:
  app:
    image: my-spring-boot-app
    ports:
      - "8080:8080"
    depends_on:
      - db

  db:
    image: postgres:latest
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
```
In this file:
- The `app` service runs the Spring Boot application.
- The `db` service runs a PostgreSQL database, with credentials passed through environment variables.
- The `depends_on` directive ensures that the database service starts before the application.

To start the services:
```bash
```

```
docker-compose up -d
```
This command starts both the application and the database in detached mode. To stop the services:
```bash
docker-compose down
```
This command stops and removes the containers.

Conclusion

Using Docker to containerize Spring Boot applications provides consistency across different environments, streamlining deployment and scaling. By leveraging Docker and Docker Compose, you can efficiently manage both single-container and multi-container applications in a DevOps setup.