**Docker Compose**

Here’s how to set up **Docker Compose** for 3 or 4 Spring Boot services, an **API Gateway**, **Spring Security**, and **MySQL**. This architecture enables secure microservice communication with centralized routing and a shared database.

**Architecture Overview**

* **Spring Boot Services**: Individual services performing different business logic.
* **API Gateway**: Handles centralized routing and security.
* **Spring Security**: Secures API Gateway and routes.
* **MySQL Database**: Shared by services for persistence.

**Step 1: Create a Docker Compose File**

version: '3.8'

services:

mysql:

image: mysql:latest

container\_name: mysql-container

networks:

- spring-network

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: mydatabase

ports:

- "3306:3306"

spring-app-1:

build:

context: ./spring-app-1

container\_name: spring-app-1

networks:

- spring-network

depends\_on:

- mysql

environment:

SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql-container:3306/mydatabase

SPRING\_DATASOURCE\_USERNAME: root

SPRING\_DATASOURCE\_PASSWORD: rootpassword

ports:

- "8081:8080"

spring-app-2:

build:

context: ./spring-app-2

container\_name: spring-app-2

networks:

- spring-network

depends\_on:

- mysql

environment:

SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql-container:3306/mydatabase

SPRING\_DATASOURCE\_USERNAME: root

SPRING\_DATASOURCE\_PASSWORD: rootpassword

ports:

- "8082:8080"

spring-app-3:

build:

context: ./spring-app-3

container\_name: spring-app-3

networks:

- spring-network

depends\_on:

- mysql

environment:

SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql-container:3306/mydatabase

SPRING\_DATASOURCE\_USERNAME: root

SPRING\_DATASOURCE\_PASSWORD: rootpassword

ports:

- "8083:8080"

api-gateway:

build:

context: ./api-gateway

container\_name: api-gateway

networks:

- spring-network

depends\_on:

- spring-app-1

- spring-app-2

- spring-app-3

ports:

- "8080:8080"

environment:

SPRING\_SECURITY\_ENABLED: true

networks:

spring-network:

**Step 2: Directory Structure**

Organize your project with the following structure:

project-root/

├── docker-compose.yml

├── mysql-init-scripts/

│ ├── schema.sql

│ ├── data.sql

├── spring-app-1/

│ ├── Dockerfile

│ └── target/

│ └── spring-app-1.jar

├── spring-app-2/

│ ├── Dockerfile

│ └── target/

│ └── spring-app-2.jar

├── spring-app-3/

│ ├── Dockerfile

│ └── target/

│ └── spring-app-3.jar

├── api-gateway/

│ ├── Dockerfile

│ └── target/

│ └── api-gateway.jar

**Step 3: Dockerfiles for Spring Boot Applications**

**Example Dockerfile for Spring Boot Services**

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/spring-app-1.jar app.jar

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]

Repeat this for spring-app-2 and spring-app-3.

**Example Dockerfile for API Gateway**

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/api-gateway.jar app.jar

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]

**Step 4: Application Configuration**

**Spring Boot Applications**

Each Spring Boot application must connect to the MySQL container.

**application.yml for Spring Boot Services**:

spring:

datasource:

url: jdbc:mysql://mysql-container:3306/mydatabase

username: root

password: rootpassword

jpa:

hibernate:

ddl-auto: update

show-sql: true

**Step 5: API Gateway Configuration**

**API Gateway Routes**

The API Gateway routes requests to the backend services.

**application.yml for API Gateway**:

spring:

cloud:

gateway:

routes:

- id: spring-app-1

uri: http://spring-app-1:8080

predicates:

- Path=/app1/\*\*

- id: spring-app-2

uri: http://spring-app-2:8080

predicates:

- Path=/app2/\*\*

- id: spring-app-3

uri: http://spring-app-3:8080

predicates:

- Path=/app3/\*\*

security:

enabled: true

**Step 6: Add Security**

**Add Security to API Gateway**

1. Add the Spring Security dependency to the API Gateway:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

1. Create a security configuration class:

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/app1/\*\*", "/app2/\*\*", "/app3/\*\*").authenticated()

.and()

.httpBasic();

}

}

**Step 7: Initialize MySQL with Data**

1. Add initialization scripts to the mysql-init-scripts folder:
   * **schema.sql**:

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100)

);

* + **data.sql**:

INSERT INTO users (name, email) VALUES (‘rajesh', 'rkcp.training@gmail.com');

1. Modify the MySQL service in docker-compose.yml to include these scripts:

volumes:

- ./mysql-init-scripts:/docker-entrypoint-initdb.d

**Step 8: Build and Start Services**

1. **Build the Docker images**:

docker-compose build

1. **Start the containers**:

docker-compose up -d

**Step 9: Test the Setup**

1. **Access API Gateway**:
   * Open http://localhost:8080/app1/ to route to spring-app-1.
   * Open http://localhost:8080/app2/ to route to spring-app-2.
   * Open http://localhost:8080/app3/ to route to spring-app-3.
2. **Verify Database Connection**:
   * Use a MySQL client or application endpoints to query the database.
3. **Test Security**:
   * Use Basic Authentication to access secured endpoints via API Gateway.