# **06.Docker Compose를 활용해 2개 이상의 컨테이너 관리하기**

1. **[실습] MySQL, Redis 컨테이너 동시에 띄워보기**

#### **✅ Docker Compose로 MySQL, Redis 실행시키기**

1. compose 파일 작성하기

compose.yml

| services:  my-db:  image: mysql  environment:  MYSQL\_ROOT\_PASSWORD: pwd1234  volumes:  - ./mysql\_data:/var/lib/mysql  ports:  - 3306:3306    my-cache-server:  image: redis  ports:  - 6379:6379 |
| --- |

* 주의) YAML 문법에서는 들여쓰기가 중요하다.

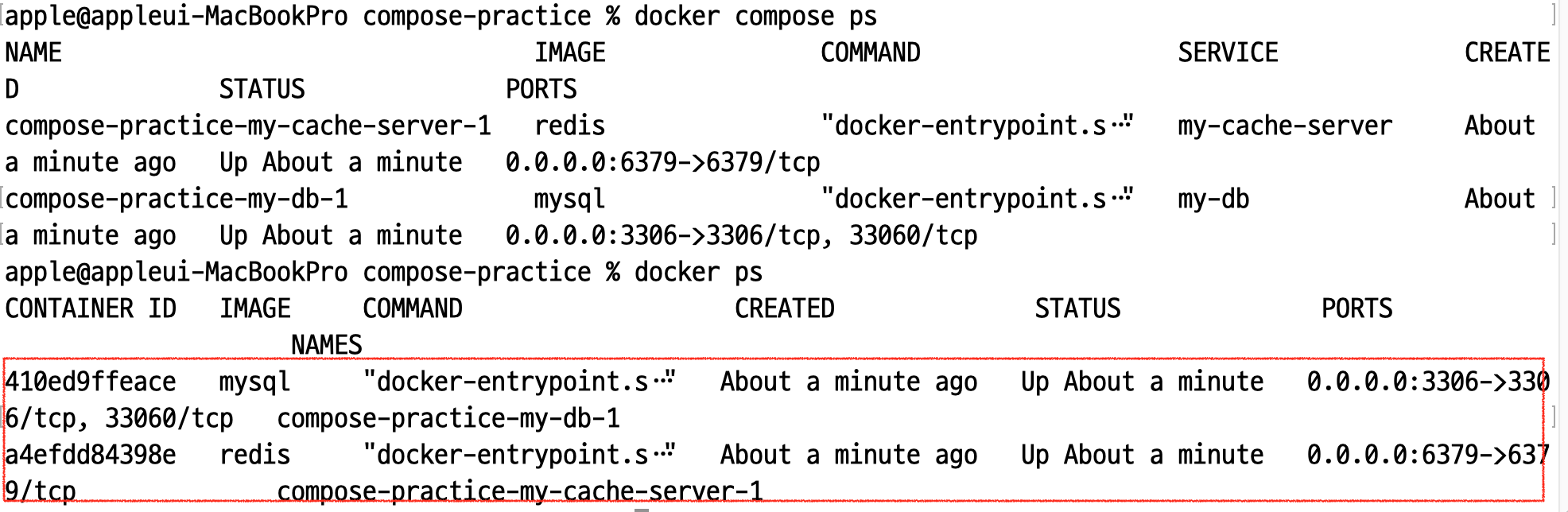
1. compose 파일 실행시키기

| $ docker compose up -d |
| --- |

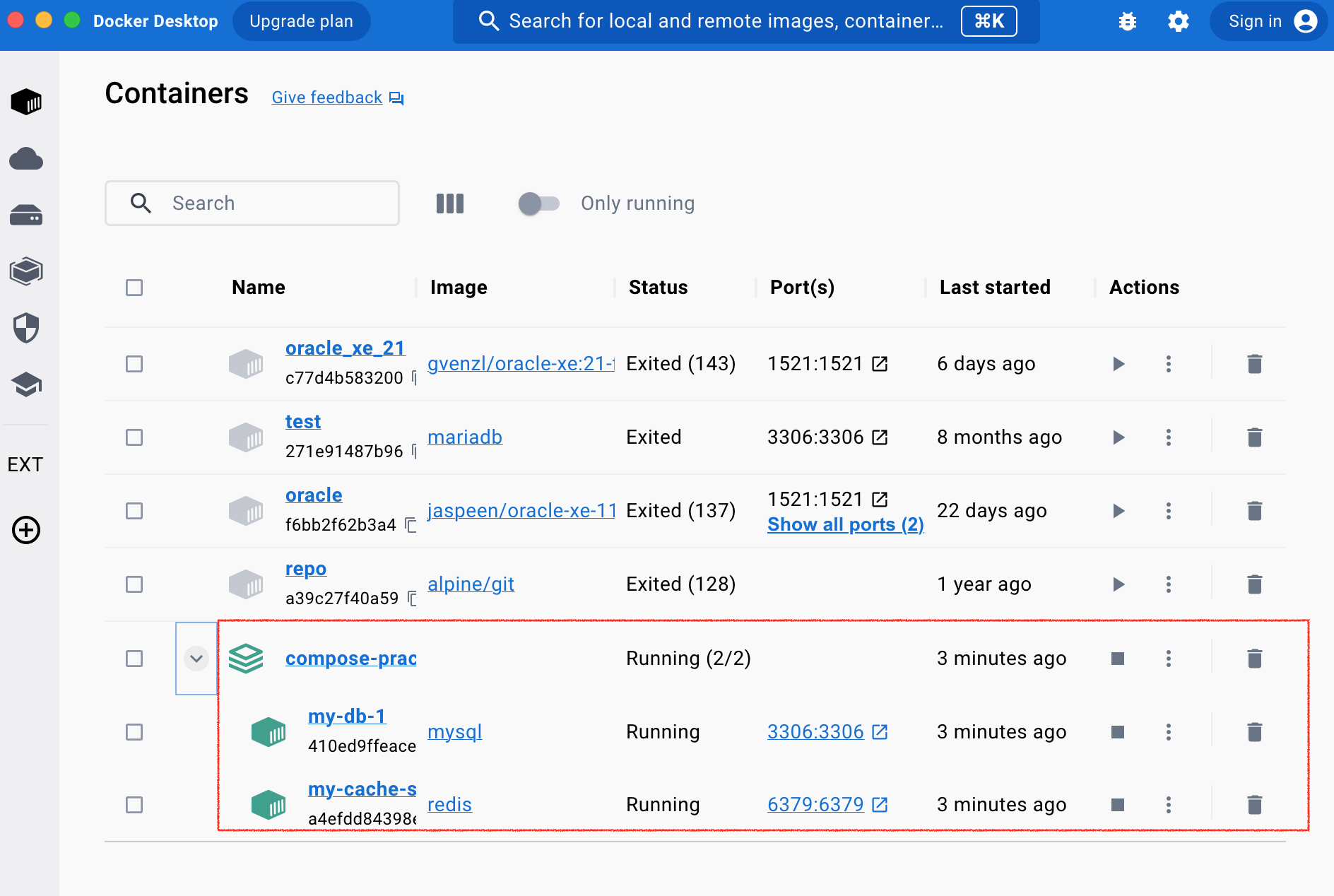


1. compose 실행 현황 보기

| $ docker compose ps  $ docker ps |
| --- |



데스크탑에서 보면



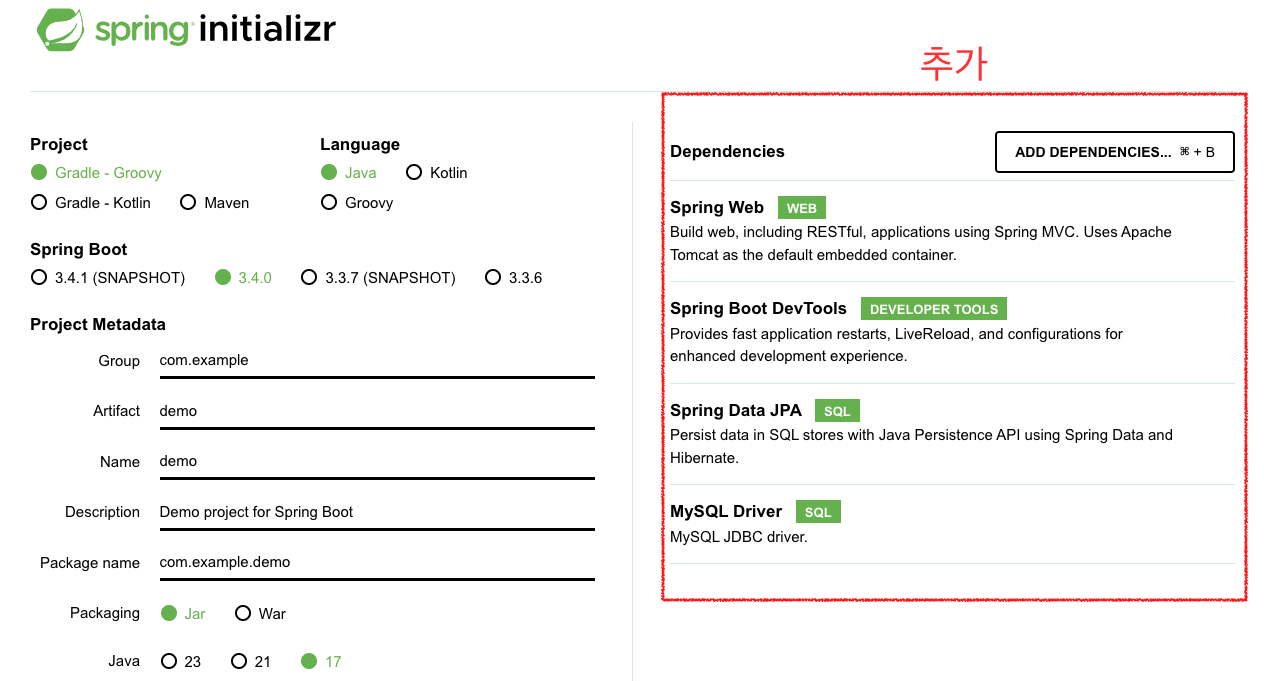
1. compose로 실행된 컨테이너 삭제

| $ docker compose down |
| --- |

<https://www.inflearn.com/community/questions/527063/compose-up-mysql-%EA%B4%80%EB%A0%A8-%EC%97%90%EB%9F%AC-%EB%9C%A8%EC%8B%9C%EB%8A%94-%EB%B6%84%EB%93%A4?srsltid=AfmBOoqYpguOvbFK-6DAvGRJgDBRAY6TVe_wX9YW9VM4ZaVHbE0ndNU4>

1. **[실습] Spring Boot, MySQL 컨테이너 동시에 띄워보기**

### **✅ Spring Boot, MySQL 컨테이너 동시에 띄워보기**

1. **Spring Boot 프로젝트 셋팅** [start.spring.io](https://start.spring.io/)  
   * Java 17 버전을 선택하자. 아래 과정을 Java 17 버전을 기준으로 진행할 예정이다.
   * Dependencies는 Spring Boot DevTools, Spring Web, Spring Data JPA, MySQL Driver를 선택해라.

| implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  implementation 'org.springframework.boot:spring-boot-starter-web'  developmentOnly 'org.springframework.boot:spring-boot-devtools'  runtimeOnly 'com.mysql:mysql-connector-j'  testImplementation 'org.springframework.boot:spring-boot-starter-test'  testRuntimeOnly 'org.junit.platform:junit-platform-launcher' |
| --- |

[**https://blog.naver.com/flutter\_java/223688777118**](https://blog.naver.com/flutter_java/223688777118)

1. **간단한 코드 작성**

AppController

| package com.example.program\_server;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.RestController;  @RestController  public class AppController {  @GetMapping("/")  public String home() {  return "Hello, World!";  }  } |
| --- |

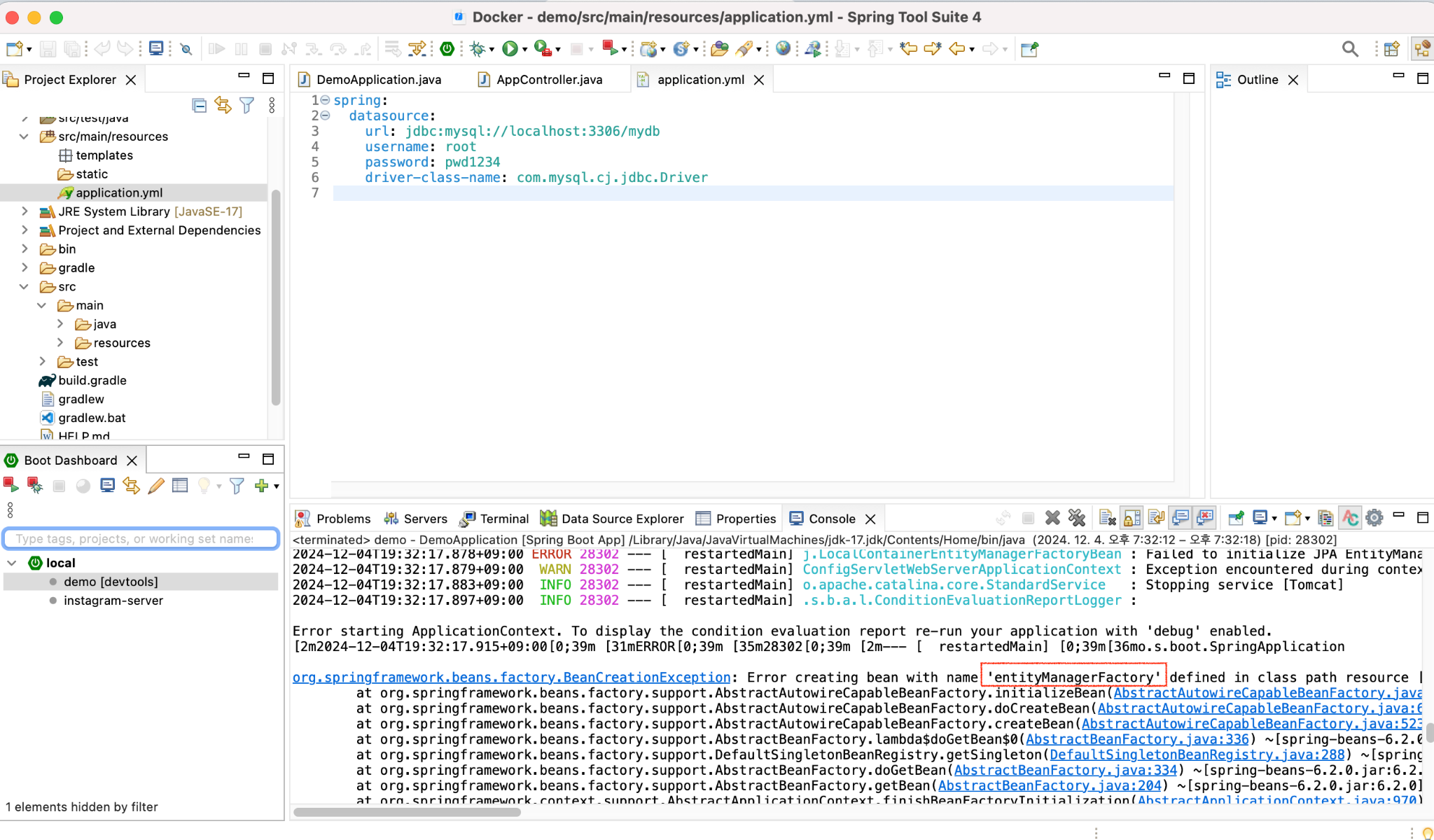
여기서 실행해 보면 DB연결에 대한 정보가 없다는 에러가 남.

1. **application.yml에 DB 연결을 위한 정보 작성하기**

**application.yml**

| spring:  datasource:  url: jdbc:mysql://localhost:3306/**mydb**  username: **root**  password: **pwd1234**  driver-class-name: com.mysql.cj.jdbc.Driver |
| --- |

여기서 다시한번 실행하면



데이터베이스가 연결이 제대로 안됐다는 에러가 뜸.

1. **불필요한 테스트 코드 삭제**



나중에 에러가 뜨는 것을 방지하기위해 테스트코드에 있는 파일을 삭제한다. 테스트 코드를 사용하려면 수정해서 사용할 것.

1. **Dockerfile 작성하기** **Dockerfile**

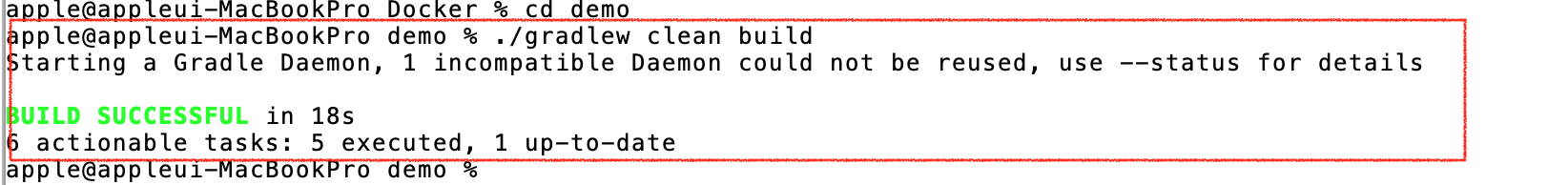
| FROM openjdk:17-jdk  COPY build/libs/\*SNAPSHOT.jar /app.jar  ENTRYPOINT ["java", "-jar", "/app.jar"] |
| --- |

1. **compose.yml 파일 작성하기**  
   **compose.yml**

| services:  my-server:  build: .  ports:  - 8080:8080  # my-db의 컨테이너가 생성되고 healthy 하다고 판단 될 때, 해당 컨테이너를 생성한다.  depends\_on:  my-db:  condition: service\_healthy  my-db:  image: mysql  environment:  MYSQL\_ROOT\_PASSWORD: pwd1234  MYSQL\_DATABASE: mydb # MySQL 최초 실행 시 mydb라는 데이터베이스를 생성해준다.  volumes:  - ./mysql\_data:/var/lib/mysql  ports:  - 3306:3306  healthcheck:  test: [ "CMD", "mysqladmin", "ping" ] # MySQL이 healthy 한 지 판단할 수 있는 명령어  interval: 5s # 5초 간격으로 체크  retries: 10 # 10번까지 재시도 |
| --- |

1. **Spring Boot 프로젝트 빌드하기**

| $ ./gradlew clean build |
| --- |



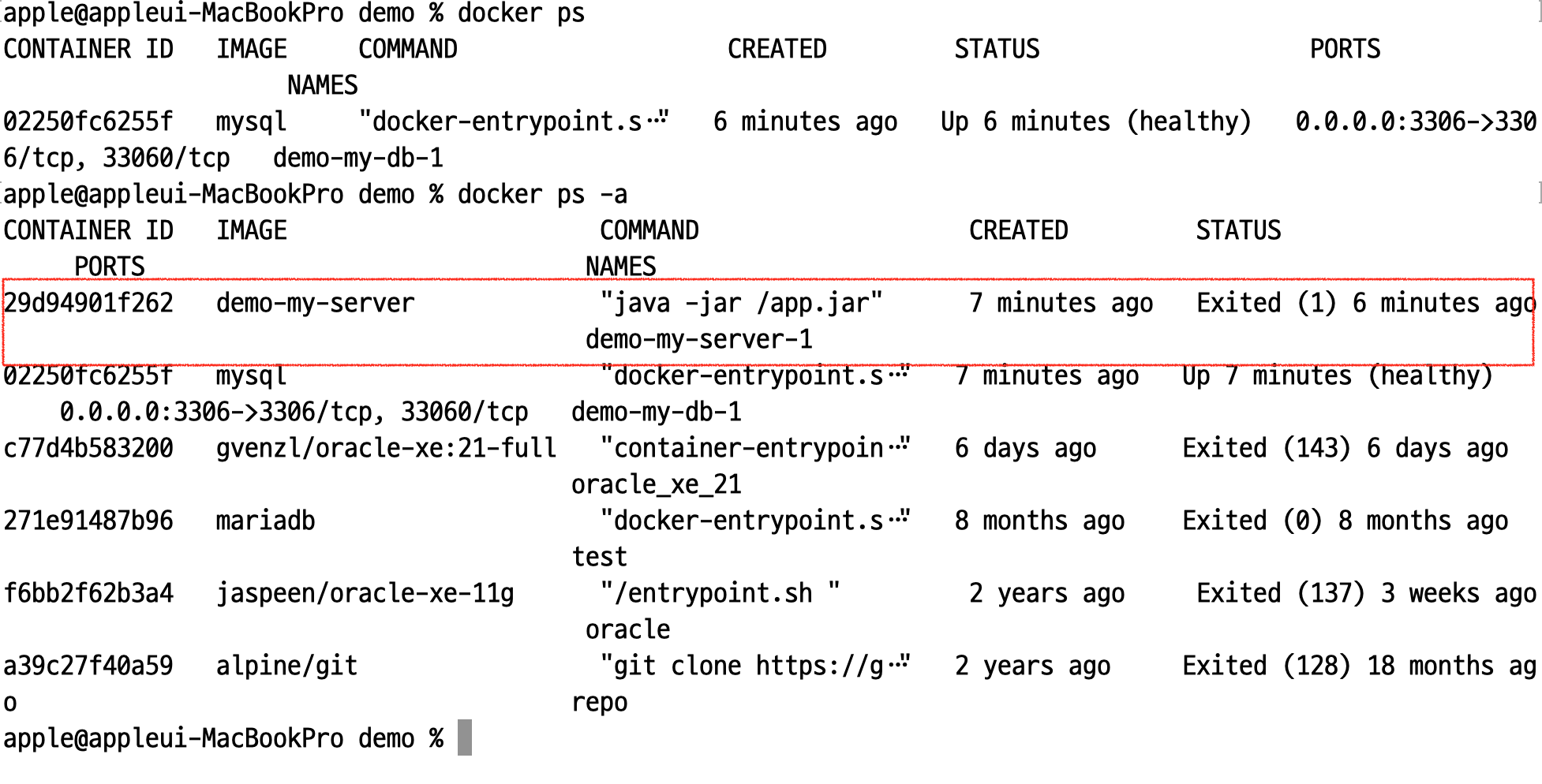
1. **compose 파일 실행시키기**

| $ docker compose up -d **--build** |
| --- |



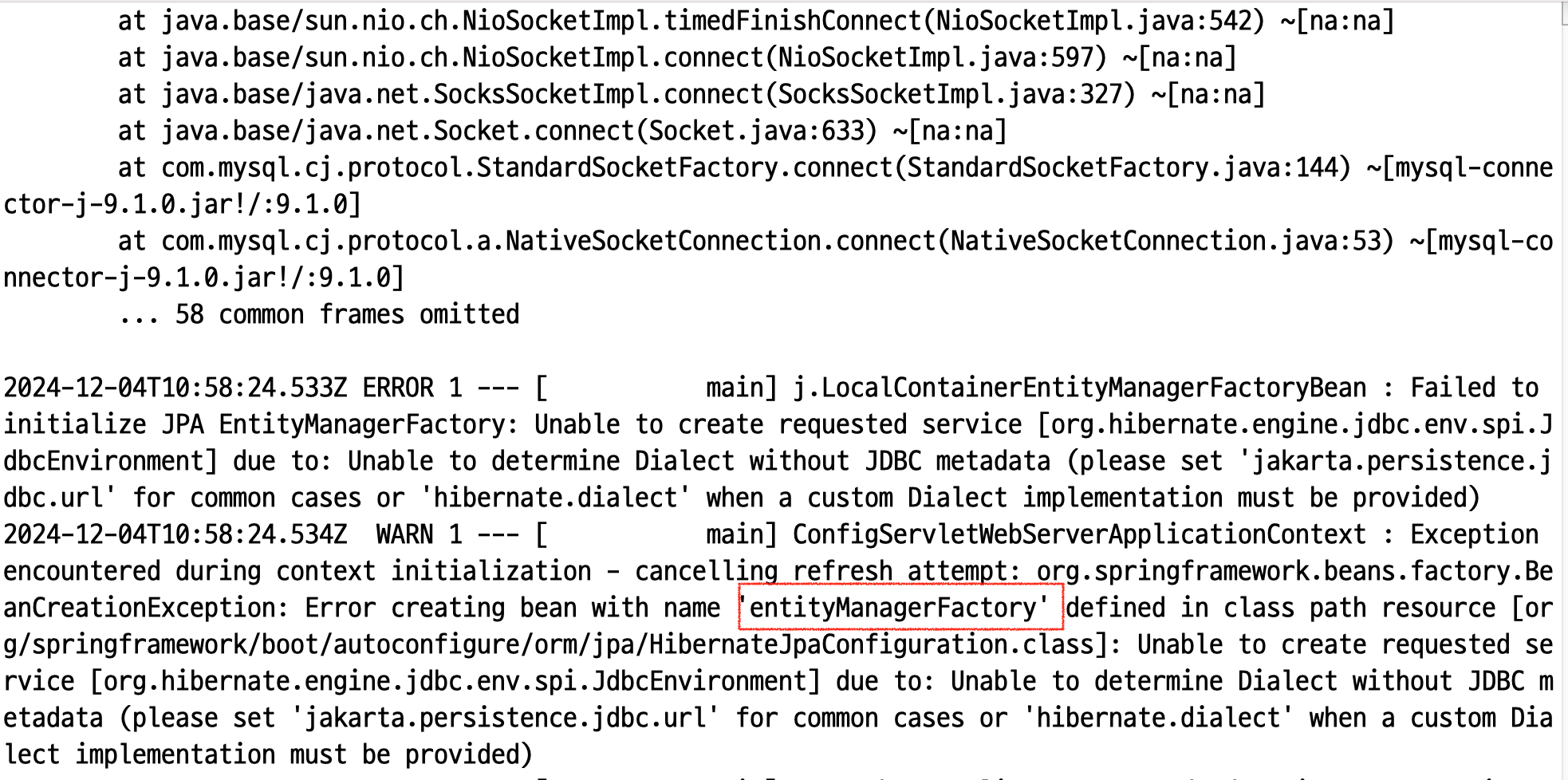
1. **compose 실행 현황 보기**

| $ docker compose ps  $ docker ps  $ docker logs [Container ID] |
| --- |

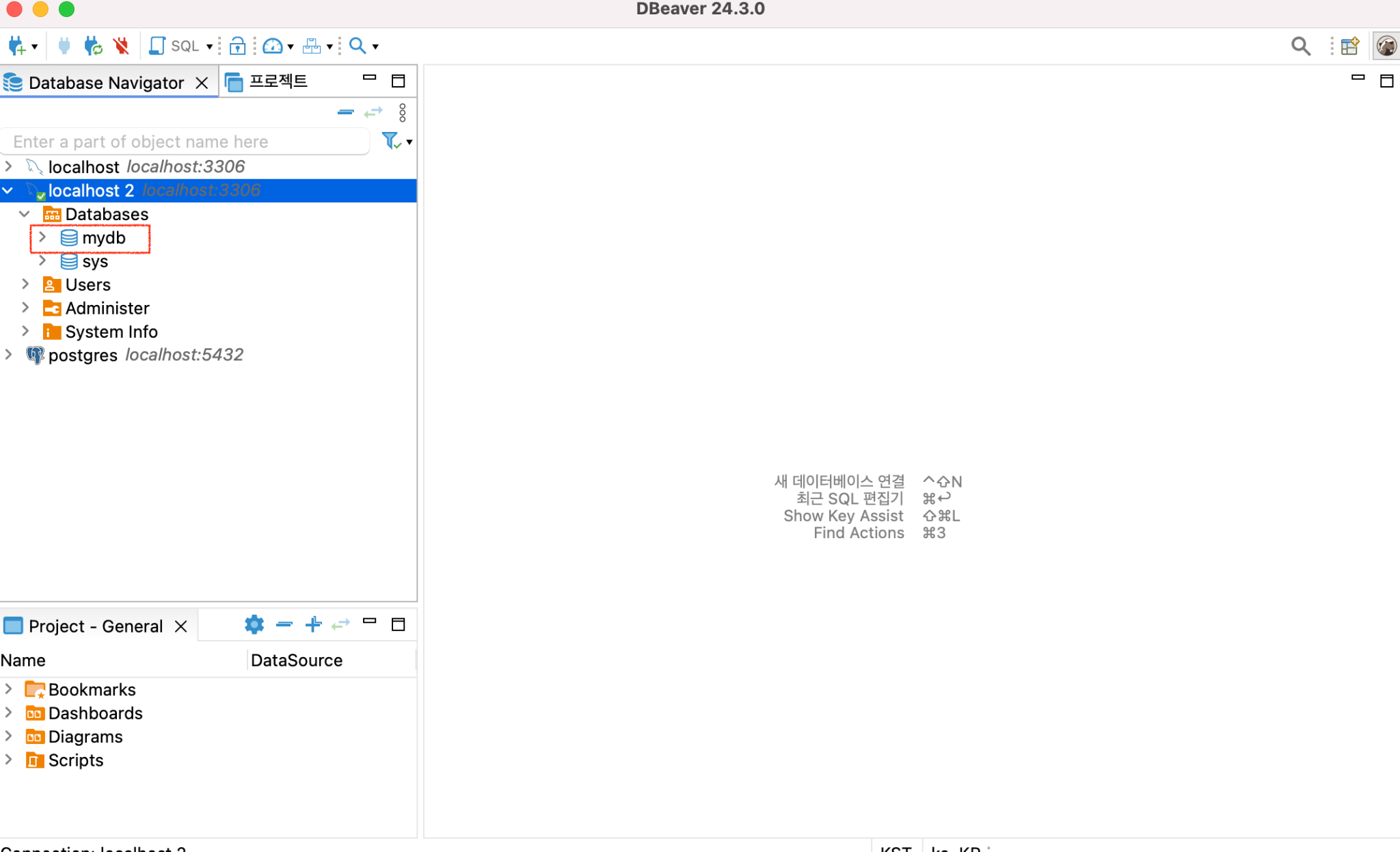


스프링 서버가 실행되지 않는 것을 볼수 있다.

* Spring Boot 컨테이너의 로그를 열어보면 아래와 같이 에러 메시지가 떠있다. 아래 에러 메시지는 DB와 연결이 제대로 이루어지지 않았을 때 발생하는 에러이다.



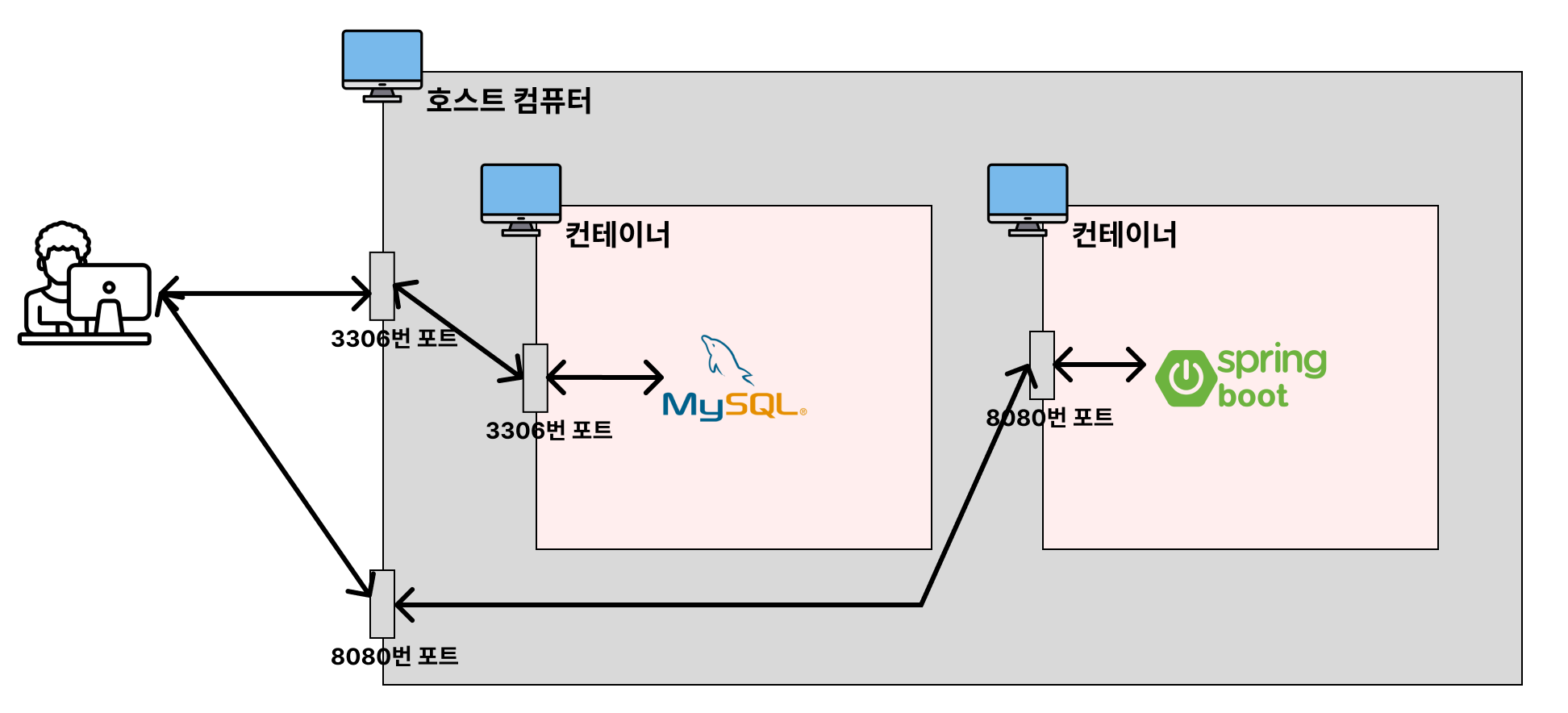
* MySQL이 정상적으로 실행이 안 되고 있는 건지 확인하기 위해 DB GUI 툴(ex. Workbench, Datagrip, DBeaver 등)을 활용해 DB 연결을 해보자. MySQL에 연결을 시도해보면 정상적으로 연결이 잘 되는 걸 확인할 수 있다. mydb도 잘 생성된 것을 볼수 있다.



❗ 그럼 도대체 무엇이 문제길래 Spring Boot가 MySQL에 연결이 안 되는 걸까? 그 원인에 대해서 다음 강의에서 알아보자.

1. **컨테이너로 실행시킨 Spring Boot가 MySQL에 연결이 안 되는 이유**

#### **✅ 컨테이너로 실행시킨 Spring Boot가 MySQL에 연결이 안 되는 이유**

****

ALT

각각의 컨테이너는 자신만의 네트워크망과 IP 주소를 가지고 있다. 호스트 컴퓨터 입장에서 localhost는 호스트 컴퓨터를 가리키지만, Spring Boot 컨테이너 입장에서 localhost는 Spring Boot 컨테이너를 가리킨다.

그런데 Spring Boot의 코드를 작성할 때 DB 정보를 아래와 같이 입력했었다. Spring Boot가 실행되는 환경인 컨테이너 입장에서 [localhost:3306](http://localhost:3306/)라는 주소는, Spring Boot 컨테이너 내부에 있는 3306번 포트와 연결을 시도하게 된다. 하지만 Spirng Boot가 실행되는 컨테이너 내부의 3306번 포트에는 아무것도 실행되고 있지 않다. 이러한 구조상의 문제 때문에 Spring Boot가 MySQL에 연결이 안 되고 있었던 것이다.

application.yml

| spring:  datasource:  url: jdbc:mysql://localhost:3306/mydb  username: root  password: pwd1234  driver-class-name: com.mysql.cj.jdbc.Driver |
| --- |

그럼 어떻게 Spring Boot의 컨테이너에서 다른 컨테이너에 존재하는 MySQL에 연결을 할 수 있을까?

compose.yml에서 정의한 Service 이름으로 서로 통신할 수 있다. 바로 예시로 알아보자.

#### **✅ Spring Boot의 DB 정보를 아래와 같이 수정한 뒤 시도해보기**

application.yml

| spring:  datasource:  url: jdbc:mysql://my-db:3306/mydb  username: root  password: pwd1234  driver-class-name: com.mysql.cj.jdbc.Driver |
| --- |

위 코드에서 my-db는 도대체 어디서 나온 값일까?

우리가 이전에 작성했던 compose.yml을 보면 각 컨테이너에 service 이름(my-server, my-db)을 작성했었다.

compose.yml

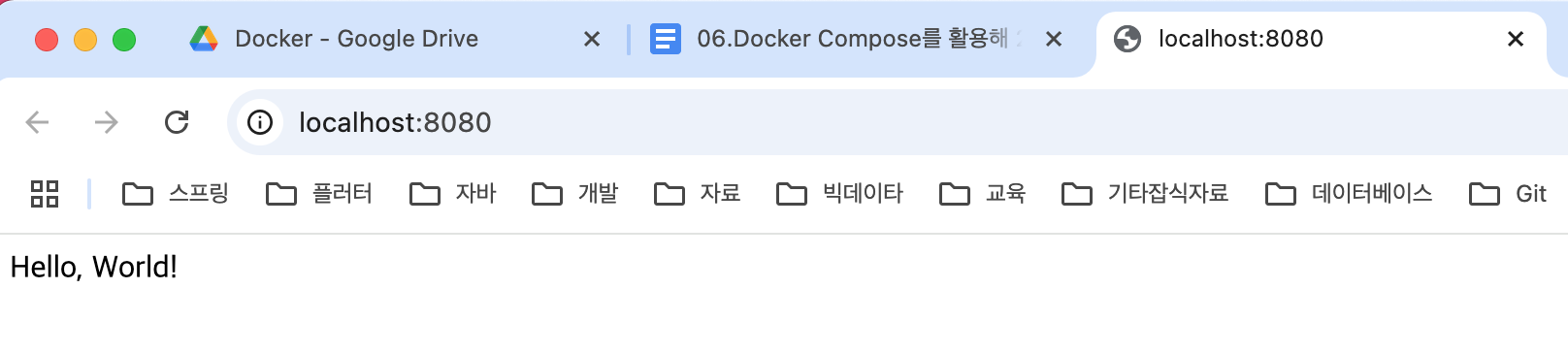
| services:  my-server:  build: .  ports:  - 8080:8080  depends\_on:  my-db:  condition: service\_healthy  my-db:  image: mysql  environment:  MYSQL\_ROOT\_PASSWORD: pwd1234  MYSQL\_DATABASE: mydb  volumes:  - ./mysql\_data:/var/lib/mysql  ports:  - 3306:3306  healthcheck:  test: [ "CMD", "mysqladmin", "ping" ]  interval: 5s  retries: 10 |
| --- |

이 service 이름이 컨테이너의 주소를 뜻한다. 해당 컨테이너의 IP 주소와 같은 역할을 한다.

위와 같이 코드를 수정한 뒤에 다시 한 번 컨테이너를 실행시켜보자.

| $ ./gradlew clean build  $ docker compose down  $ docker compose up --build -d  $ docker ps # 정상적으로 Spring Boot, MySQL이 실행된 걸 확인할 수 있다.  $ docker logs [Spring Boot 컨테이너ID] # 정상적으로 실행된다.  $ docker compose logs # 전체 로그 확인 |
| --- |

[localhost:8080](http://localhost:8080) **접속해서 확인**

****

**잘 접속 되는 것을 확인할 수 있다.**

**oracle**

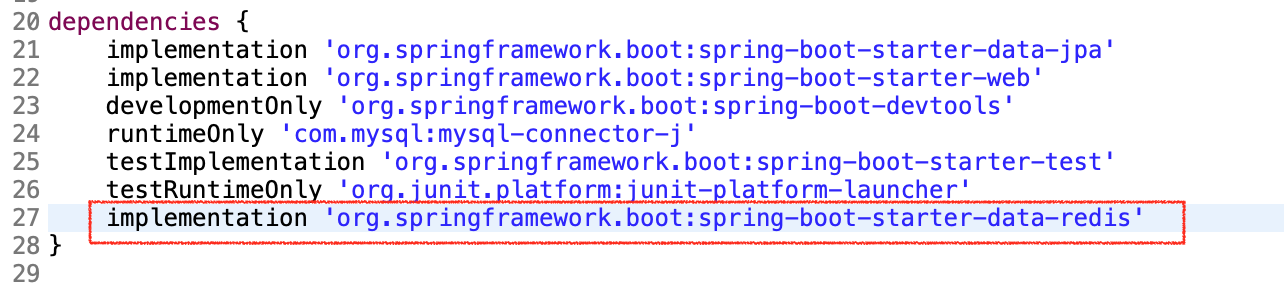
| **services:**  **oracle11g:**  **image: jaspeen/oracle-xe-11g**  **container\_name: oracle11g**  **volumes:**  **- ./jaspeen/oracle-xe-11g**  **ports:**  **- 1521:1521** |
| --- |

1. **[실습] Spring Boot, MySQL, Redis 컨테이너 동시에 띄워보기**

### **✅ 1. Spring Boot 프로젝트에 Redis 연결 코드 추가하기**

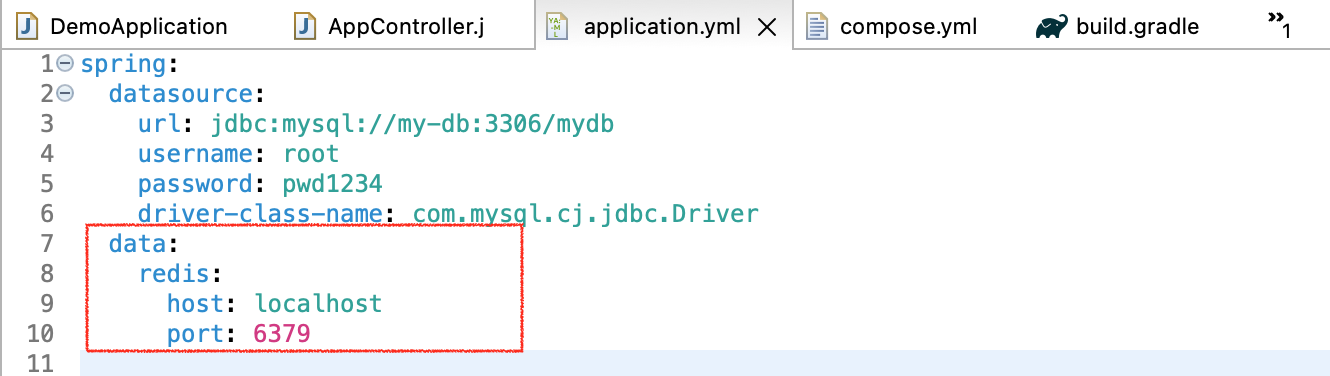
**build.gradle**

| dependencies {  ...  **implementation 'org.springframework.boot:spring-boot-starter-data-redis'**  } |
| --- |



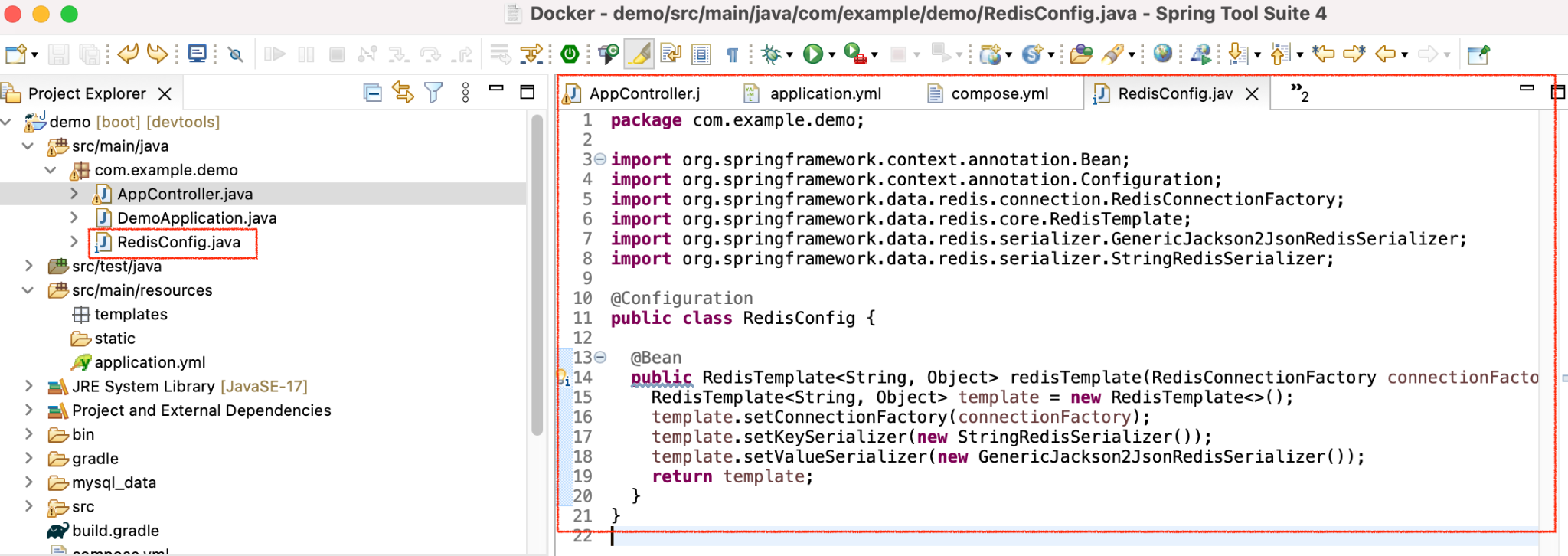
**application.yml**

| spring:  datasource:  url: jdbc:mysql://my-db:3306/mydb  username: root  password: pwd1234  driver-class-name: com.mysql.cj.jdbc.Driver  **data:**  **redis:**  **host: localhost**  **port: 6379** |
| --- |



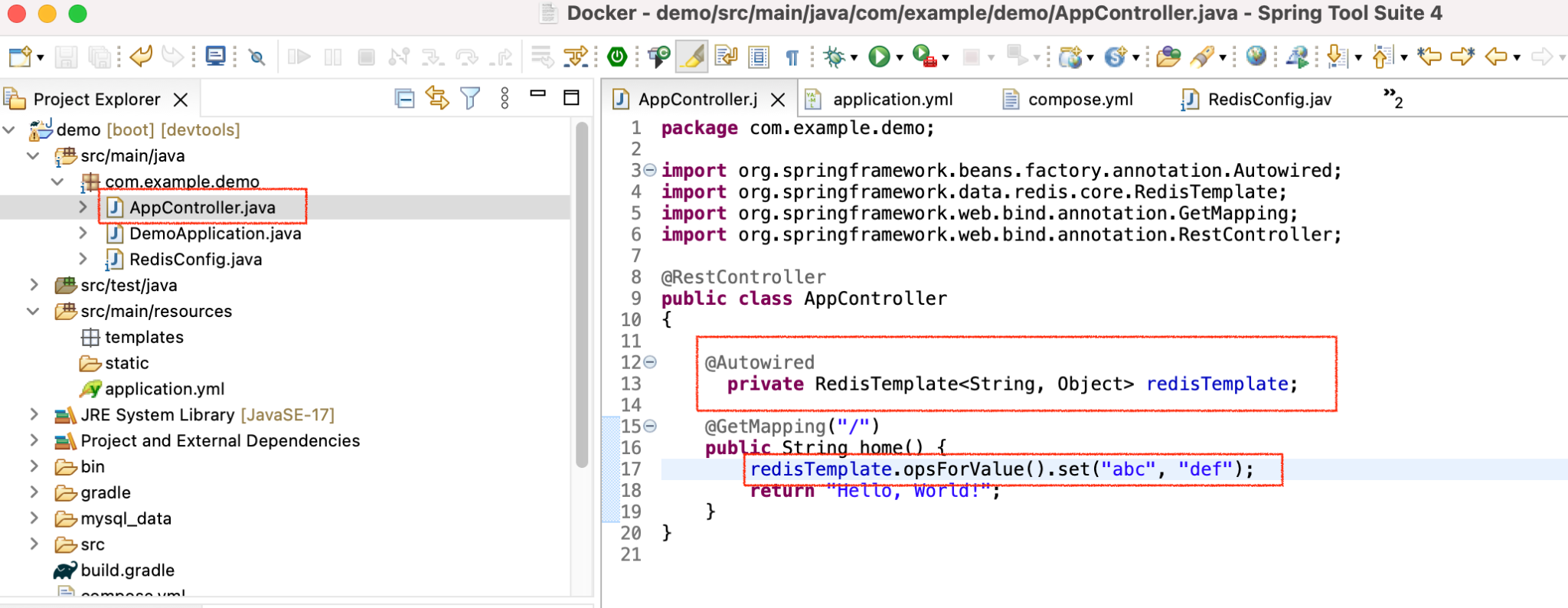
**RedisConfig**

| @Configuration  public class RedisConfig {  @Bean  public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory connectionFactory) {  RedisTemplate<String, Object> template = new RedisTemplate<>();  template.setConnectionFactory(connectionFactory);  template.setKeySerializer(new StringRedisSerializer());  template.setValueSerializer(new GenericJackson2JsonRedisSerializer());  return template;  }  } |
| --- |



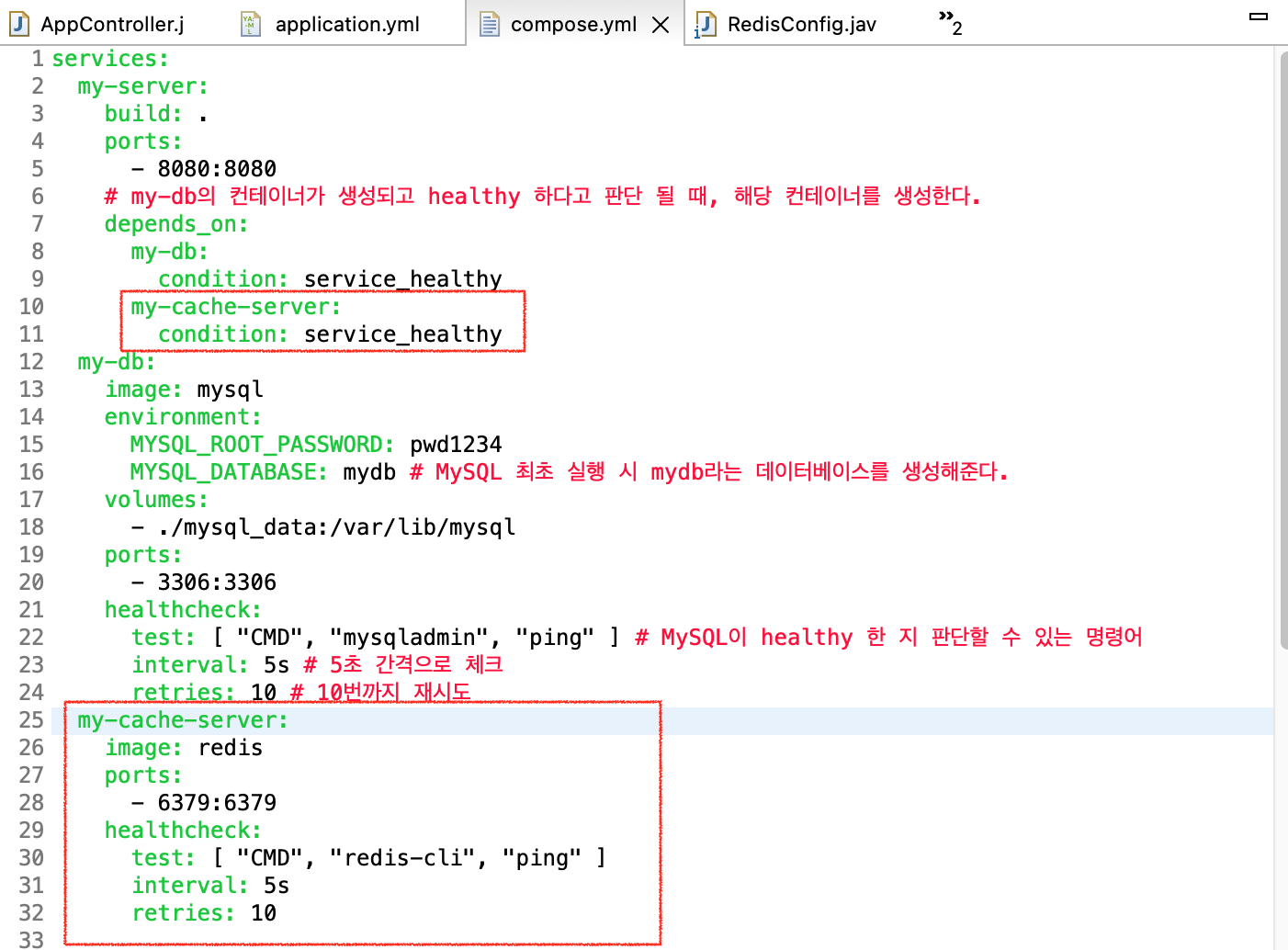
**AppController**

| @RestController  public class AppController {  **@Autowired**  **private RedisTemplate<String, Object> redisTemplate;**  @GetMapping("/")  public String home() {  **redisTemplate.opsForValue().set("abc", "def");**  return "Hello, World!";  }  } |
| --- |

****

**compose.yml**

| services:  my-server:  build: .  ports:  - 8080:8080  depends\_on:  my-db:  condition: service\_healthy  **my-cache-server:**  **condition: service\_healthy**  my-db:  image: mysql  environment:  MYSQL\_ROOT\_PASSWORD: pwd1234  MYSQL\_DATABASE: mydb  volumes:  - ./mysql\_data:/var/lib/mysql  ports:  - 3306:3306  healthcheck:  test: [ "CMD", "mysqladmin", "ping" ]  interval: 5s  retries: 10  **my-cache-server:**  **image: redis**  **ports:**  **- 6379:6379**  **healthcheck:**  **test: [ "CMD", "redis-cli", "ping" ]**  **interval: 5s**  **retries: 10** |
| --- |

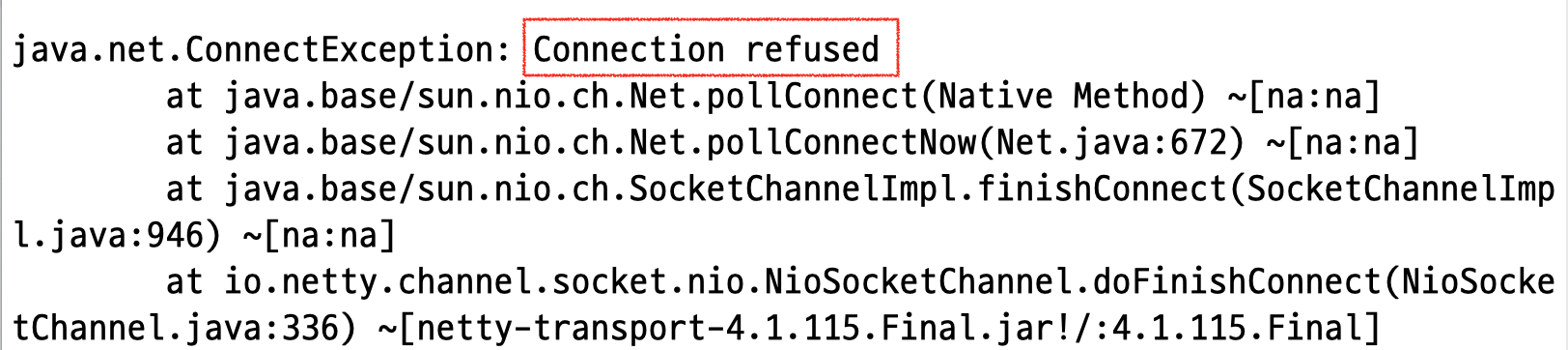
****

### **✅ 2. Docker 컨테이너로 띄워보기**

| $ ./gradlew clean build  $ docker compose down  $ docker compose up --build -d |
| --- |

위 명령어를 통해 컨테이너를 띄운 뒤에 [localhost:8080](http://localhost:8080)으로 요청을 해보면 아래와 같은 에러가 발생한다.

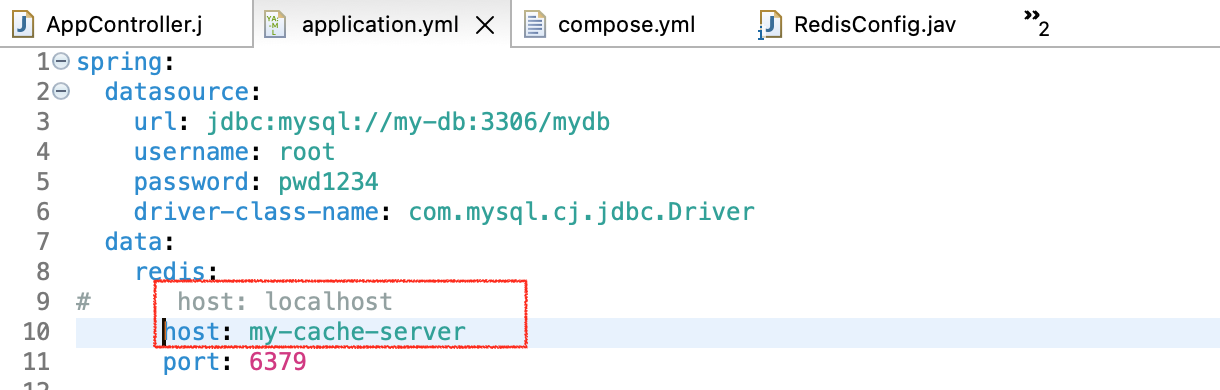




Connection refused 에러가 발생한 이유는 Redis와 연결이 잘 안 됐기 때문이다. 왜 안됐는 지 application.yml 파일을 다시 한 번 살펴보자.

**application.yml**

| spring:  datasource:  url: jdbc:mysql://my-db:3306/mydb  username: root  password: pwd1234  driver-class-name: com.mysql.cj.jdbc.Driver  data:  redis:  **~~host: localhost~~**  **host: my-cache-server**  port: 6379 |
| --- |

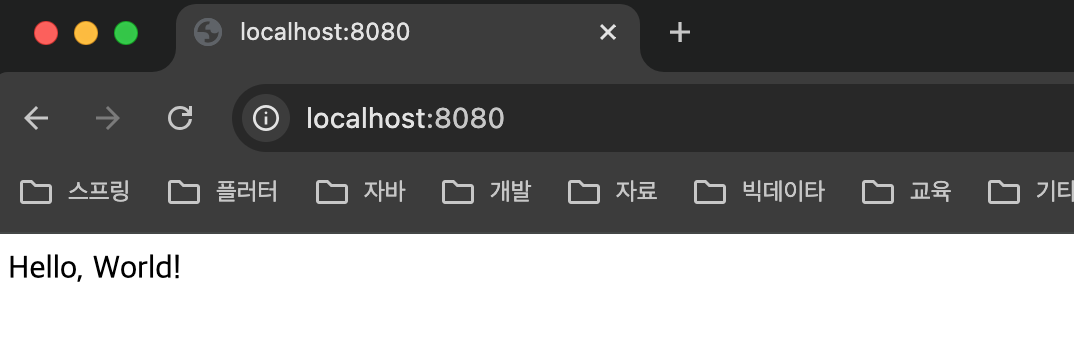
****

각 컨테이너는 각자의 네트워크를 가지고 있기 때문에, localhost가 아니라 Redis가 실행되고 있는 컨테이너로 통신을 해야 한다. Redis가 실행되고 있는 컨테이너의 주소는 service 이름으로 표현한다고 했다. compose.yml에서 Redis가 실행되고 있는 컨테이너의 service 이름을 my-cache-server라고 이름 붙였다.

위와 같이 코드를 수정한 뒤에 다시 한 번 실행시켜보자.

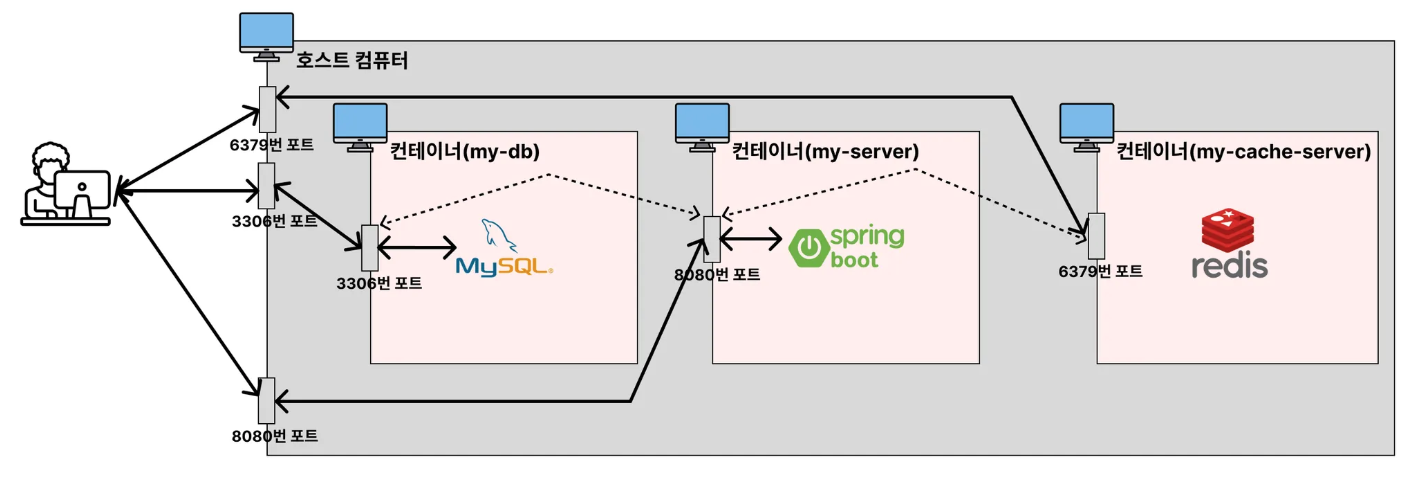
| $ ./gradlew clean build  $ docker compose down  $ docker compose up --build -d |
| --- |

[localhost:8080](http://localhost:8080)



에러가 발생하지 않고 정상적으로 실행되는 걸 확인할 수 있다.

### **✅ 그림으로 이해하기**

****