# **05.볼륨(Volume)을 활용해 DB 띄워보기**

1. **볼륨(Volume)이란?**

### **✅ 파드(Pod)가 가진 문제점**

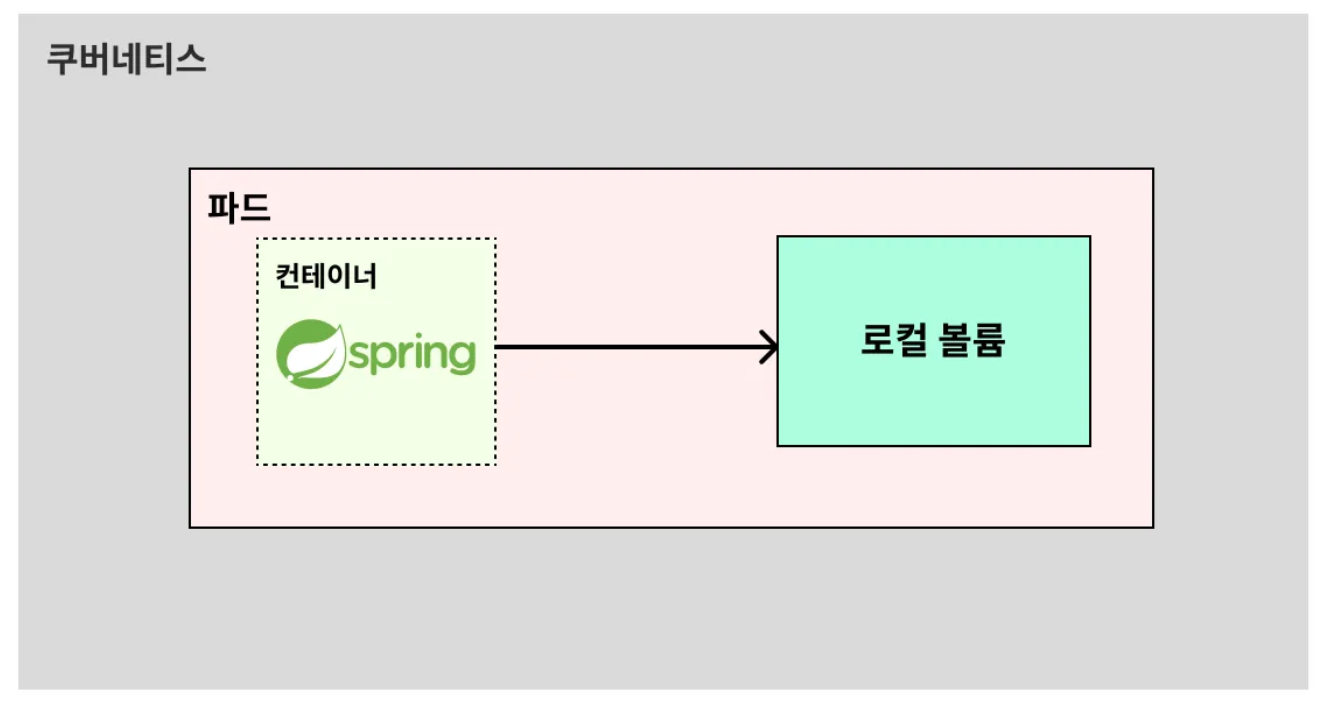
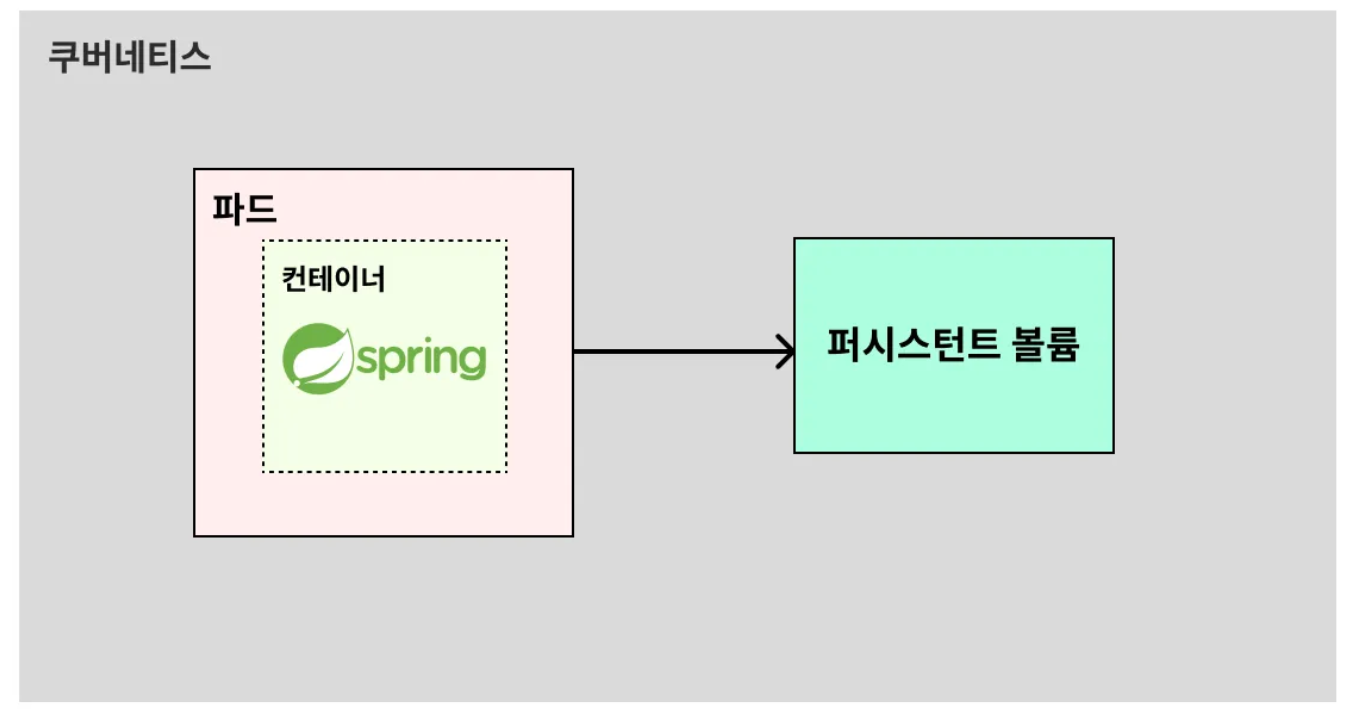
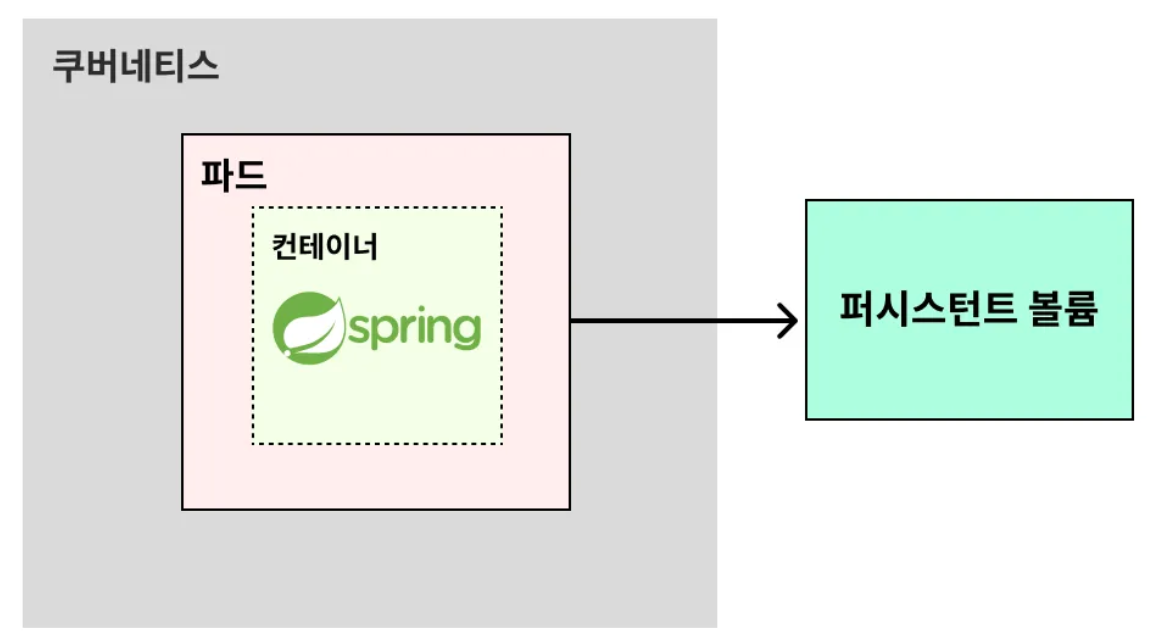
쿠버네티스를 활용하면 특정 프로그램을 파드로 띄울 수 있다. 이 프로그램에 기능이 추가되면 쿠버네티스는 기존 파드에서 변경된 부분을 수정하지 않고, 새로운 파드를 만들어서 통째로 갈아끼우는 방식으로 교체를 한다. 이게 효율적이라고 생각했던 것이다.

이런 특징 때문에 기존 파드를 새로운 파드로 교체하면, 기존 파드 내부에 있던 데이터도 같이 삭제된다. 만약 이 파드가 MySQL을 실행시키는 파드였다면 MySQL에 저장된 데이터도 같이 삭제 돼버린다.

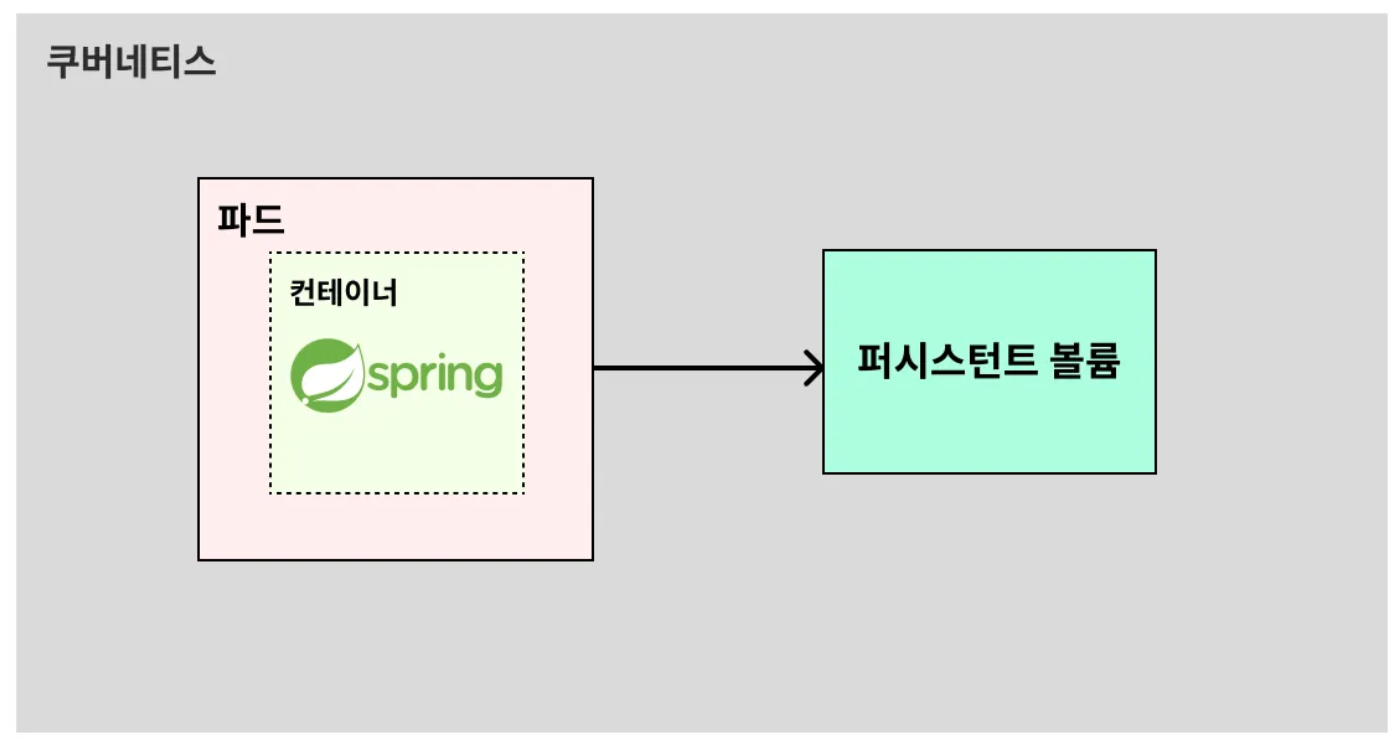
따라서 파드 내부에 저장된 데이터가 삭제되면 안 되는 경우에는 **볼륨(Volume)**이라는 개념을 활용해야 한다.

### **✅ 쿠버네티스의 볼륨(Volume)이란?**

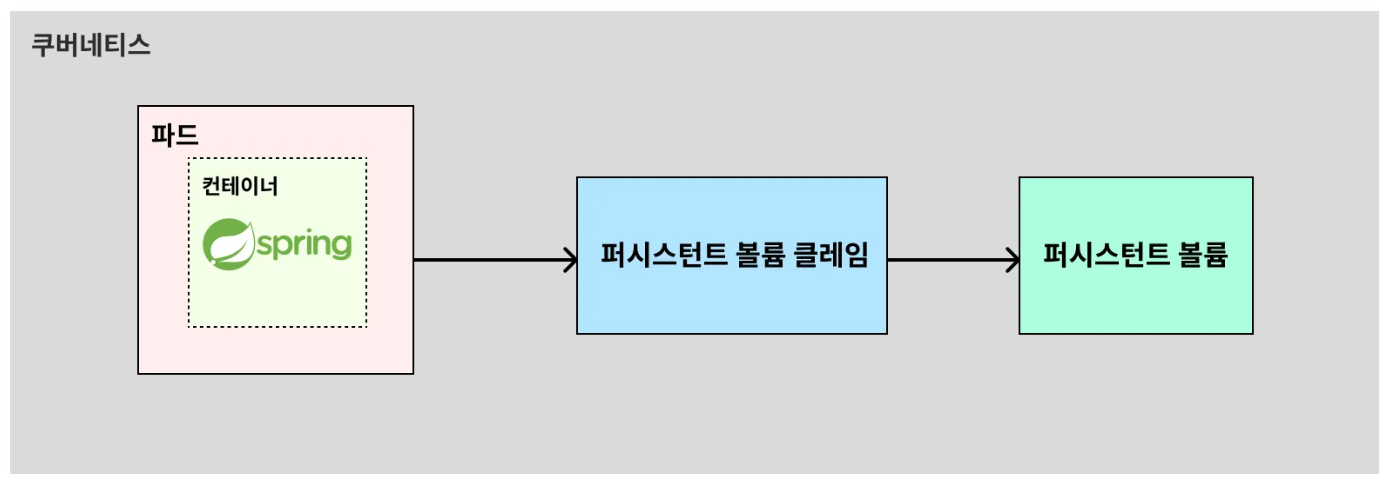
**볼륨(Volume)**이란 **데이터를 영속적으로 저장하기 위한 방법**이다. 쿠버네티스에서 볼륨은 크게 2가지 종류로 나뉜다.

1. **로컬 볼륨**  
     
    **파드 내부의 공간 일부를 볼륨(Volume)으로 활용하는 방식**이다. 이 방식은 파드가 삭제되는 즉시 데이터도 함께 삭제된다. 이런 불편함 때문에 실제로 사용되는 일이 잘 없다.  
    
2. **퍼시스턴트 볼륨(Persistent Volume, PV)**  
     
    **파드 외부의 공간 일부를 볼륨(Volume)**으로 활용하는 방식이다. 이 방식은 파드가 삭제되는 것과 상관없이 데이터를 영구적으로 사용할 수 있다는 장점이 있다. 현업에서는 주로 이 방식을 많이 활용한다.   
    쿠버네티스 내부의 공간 일부를 사용하는 경우  
     
      
    외부 저장소(AWS EBS 등)를 사용하는 경우

### **✅ 퍼시스턴트 볼륨 클레임(Persistent Volume Claim, PVC)이란?**

****

퍼시스턴트 볼륨(PV)을 설명할 때 위와 같은 그림으로 설명했다. 실제로는 파드(Pod)가 퍼시스턴트 볼륨(PV)에 직접 연결할 수 없다. 퍼시스턴트 볼륨 클레임(PVC)이라는 중개자가 있어야 한다. 그래서 쿠버네티스 구조에서는 아래와 같은 구조로 퍼시스턴트 볼륨(PV)을 연결한다.

****

퍼시스턴트 볼륨 클레임(PVC)은 파드(Pod)와 퍼시스턴트 볼륨(PV) 사이에서 중개자 역할을 한다.

1. **[예제] 디플로이먼트(Deployment)를 활용해 MySQL 실행시키기**

### **✅ 디플로이먼트(Deployment) 활용해 MySQL 실행시키기**

1. **매니페스트 파일 생성하기**

mysql-deployment.yaml

| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: mysql-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: **1** # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: mysql-db # 아래에서 정의한 Pod 중 'app: mysql-db'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: mysql-db  spec:  containers:  - name: mysql-container # 컨테이너 이름  image: mysql # 컨테이너를 생성할 때 사용할 이미지  ports:  - containerPort: 3306 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  - name: MYSQL\_ROOT\_PASSWORD  ~~value: password123~~  valueFrom:  secretKeyRef:  name: mysql-secret  key: mysql-root-password  - name: MYSQL\_DATABASE  ~~value: kub-practice~~  valueFrom:  configMapKeyRef:  name: mysql-config  key: mysql-database |
| --- |

mysql-secret.yaml

| apiVersion: v1  kind: Secret  # Secret 기본 정보  metadata:  name: mysql-secret # Secret 이름  # Key, Value 형식으로 값 저장  stringData:  mysql-root-password: password123 |
| --- |

mysql-config.yaml

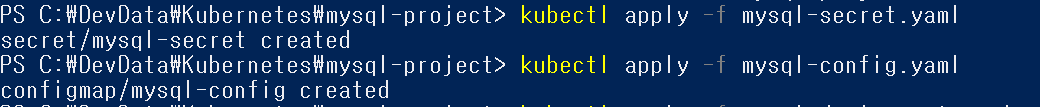
| apiVersion: v1  kind: ConfigMap  # ConfigMap 기본 정보  metadata:  name: mysql-config # ConfigMap 이름  # Key, Value 형식으로 설정값 저장  data:  mysql-database: kub-practice  mysql-service.yaml |
| --- |

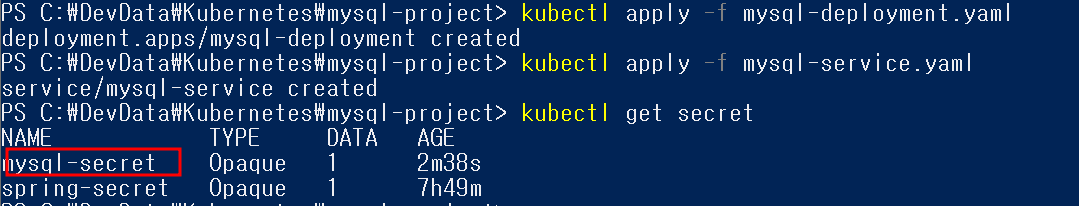
mysql-service.yaml

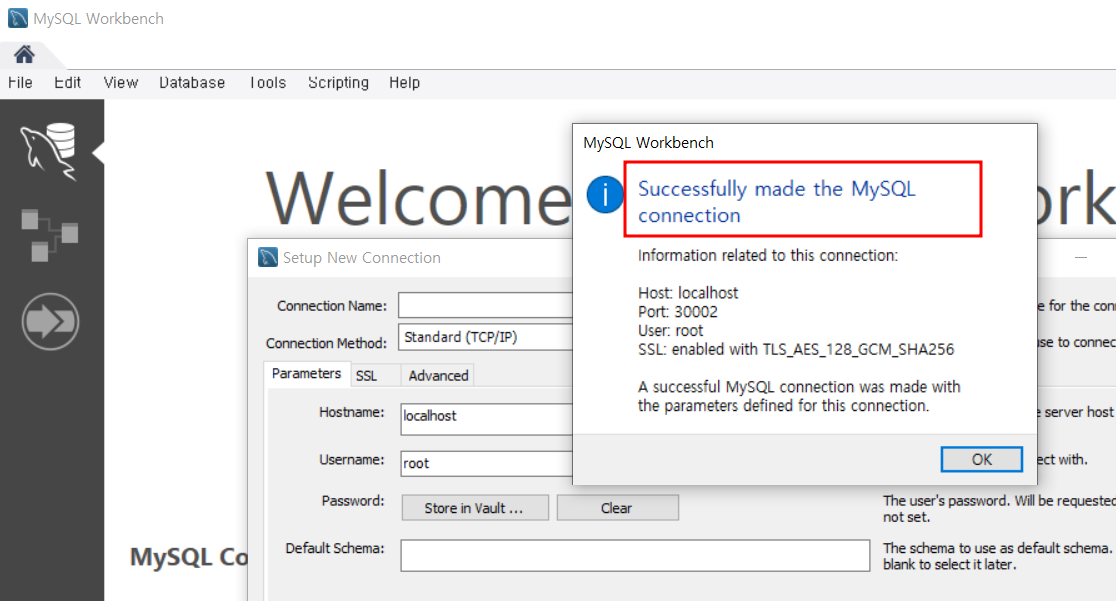
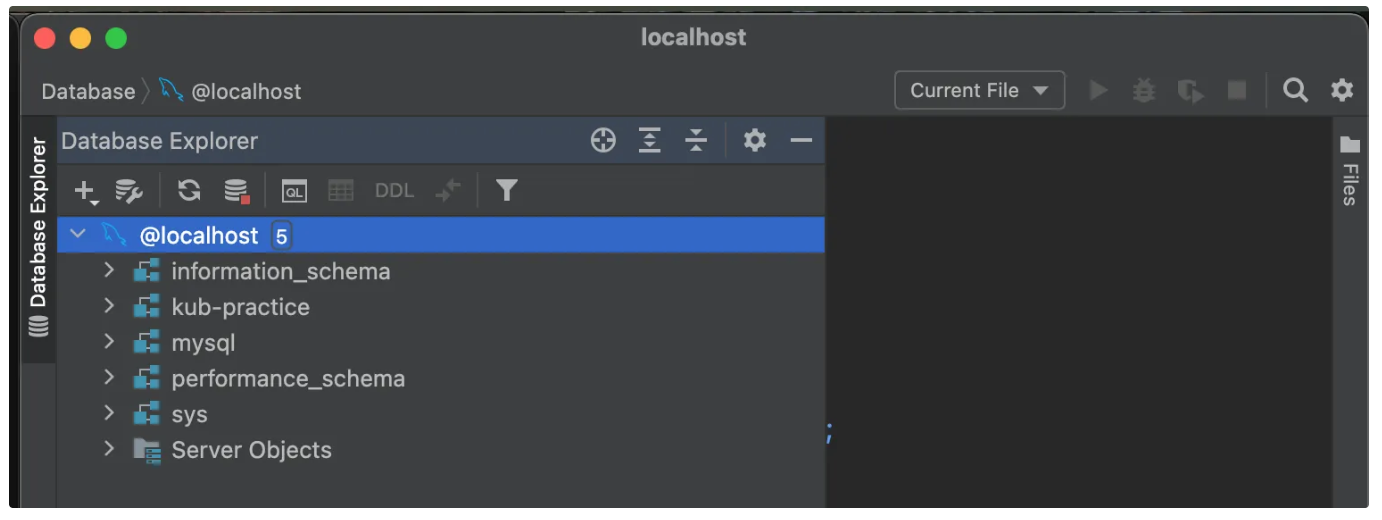
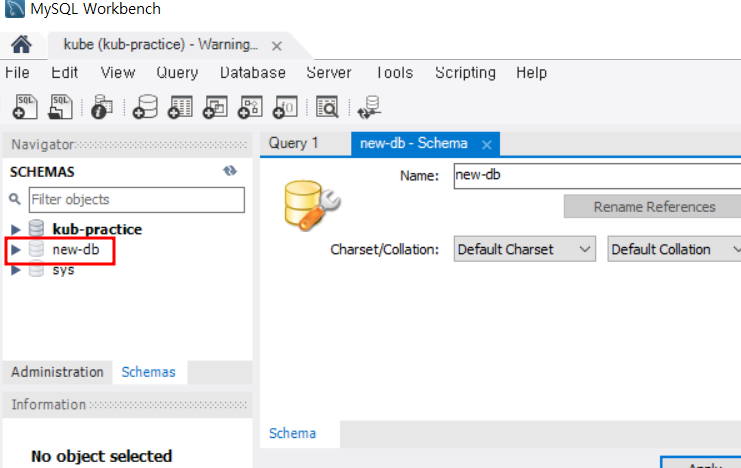
| apiVersion: v1  kind: Service  # Service 기본 정보  metadata:  name: mysql-service # Service 이름    # Service 세부 정보  spec:  type: NodePort # Service의 종류  selector:  app: mysql-db # 실행되고 있는 파드 중 'app: mysql-db'이라는 값을 가진 파드와 서비스를 연결  ports:  - protocol: TCP # 서비스에 접속하기 위한 프로토콜  port: 3306 # 쿠버네티스 내부에서 Service에 접속하기 위한 포트 번호  targetPort: 3306 # 매핑하기 위한 파드의 포트 번호  nodePort: 30002 # 외부에서 사용자들이 접근하게 될 포트 번호 |
| --- |

1. 매니페스트 기반으로 오브젝트 생성하기

| $ kubectl apply -f mysql-secret.yaml  $ kubectl apply -f mysql-config.yaml  $ kubectl apply -f mysql-deployment.yaml  $ kubectl apply -f mysql-service.yaml |
| --- |





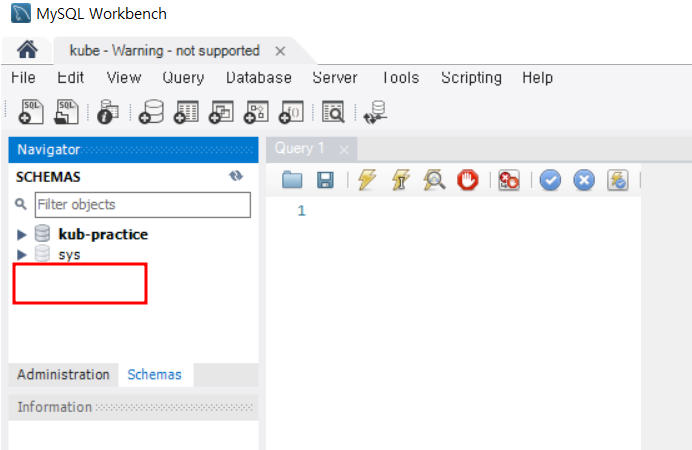
1. DB 접속해보기  
     
   
2. 새로운 데이터베이스 생성하기  
   

### ✅ MySQL 재시작시키기

🧑여기서 MySQL가 이유 모를 장애로 서비스가 중단됐다고 가정하자. 그래서 디플로이먼트를 다시 시작해야 하는 상황이라고 해보자.

1. 디플로이먼트 재시작하기

| $ kubectl rollout restart deployment mysql-deployment |
| --- |



1. 데이터베이스 다시 확인해보기  
     
    데이터베이스를 확인해보면 이전에 만들었던 **new-db**라는 데이터베이스가 없어졌다. 실제 데이터베이스에서는 데이터가 없어져서는 안 된다. 데이터가 보존되도록 볼륨(Volume)을 설정해보자.
2. **볼륨(Volume)을 활용해 MySQL 실행시키기**

### **✅ 볼륨(Volume)을 활용해 MySQL 실행시키기**

🧑데이터베이스를 확인해보면 이전에 만들었던 **new-db**라는 데이터베이스가 없어졌다. 실제 데이터베이스에서는 데이터가 없어져서는 안 된다. 데이터가 보존되도록 볼륨(Volume)을 설정해보자.

1. **퍼시스턴트 볼륨(PV), 퍼시스턴트 볼륨 클레임(PVC) 정의하기**

mysql-pv.yaml

| apiVersion: v1  kind: PersistentVolume  # PersistentVolume 기본 정보  metadata:  name: mysql-pv # PersistentVolume 이름  # PersistentVolume 세부 정보  spec:  storageClassName: my-storage # PV와 PVC의 storageClassName이 같다면 볼륨이 연결된다.  capacity:  storage: 1Gi # 볼륨이 사용할 용량을 설정  accessModes:  - ReadWriteOnce # 아래 hostPath 타입 활용 시 이 옵션만 사용 가능  hostPath: # hostPath 타입을 활용 (hostPath : 쿠버네티스 내부 공간을 활용)  path: "/mnt/data" # 쿠버네티스 내부의 공간에서 /mnt/data의 경로를 볼륨으로 사용 |
| --- |

mysql-pvc.yaml

| apiVersion: v1  kind: PersistentVolumeClaim  # PersistentVolumeClaim 기본 정보  metadata:  name: mysql-pvc # PersistentVolumeClaim 이름    # PersistentVolumeClaim 세부 정보  spec:  storageClassName: my-storage # PV와 PVC의 storageClassName이 같다면 볼륨이 연결된다.  accessModes:  - ReadWriteOnce # 볼륨에 접근할 때의 권한  resources: # PVC가 PV에 요청하는 리소스의 양을 정의  requests: # 필요한 최소 리소스  storage: 1Gi # PVC가 PV에 요청하는 스토리지 양 (PV가 최소 1Gi 이상은 되어야 한다.) |
| --- |

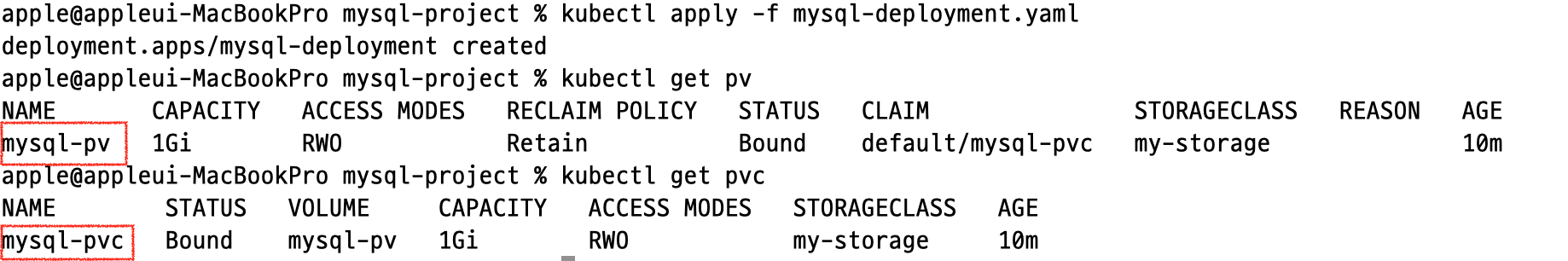
1. **기존 디플로이먼트(Deployment) 수정하기**

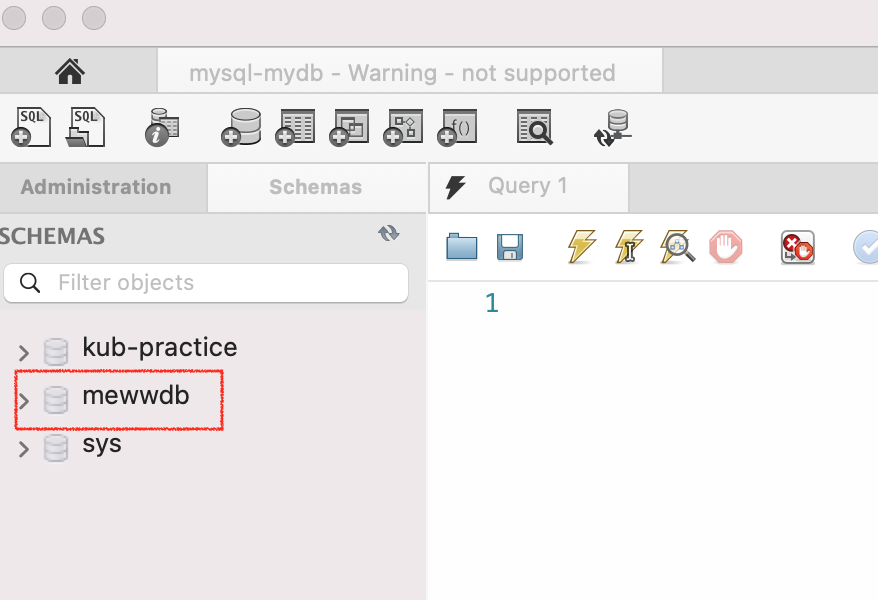
mysql-deploymentyaml.yaml

| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: mysql-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 1 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: mysql-db # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: mysql-db  spec:  containers:  - name: mysql-container # 컨테이너 이름  image: mysql # 컨테이너를 생성할 때 사용할 이미지  ports:  - containerPort: 3306 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  - name: MYSQL\_ROOT\_PASSWORD  valueFrom:  secretKeyRef:  name: mysql-secret  key: mysql-root-password  - name: MYSQL\_DATABASE  valueFrom:  configMapKeyRef:  name: mysql-config  key: mysql-database  # 컨테이너 내에서 어떤 경로를 볼륨으로 사용할 지 지정  **volumeMounts:**  **- name: mysql-persistent-storage** # 밑에서 설정할 volumes.name과 값이 같아야 함  **mountPath: /var/lib/mysql** # mysql 컨테이너 내부에 있는 경로  # 파드가 사용할 볼륨을 지정  **volumes:**  **- name: mysql-persistent-storage** # 위에서 설정할 volumeMounts.name과 일치해야 함  **persistentVolumeClaim:**  **claimName: mysql-pvc** # 연결시킬 PVC의 name과 동일해야 함 |
| --- |

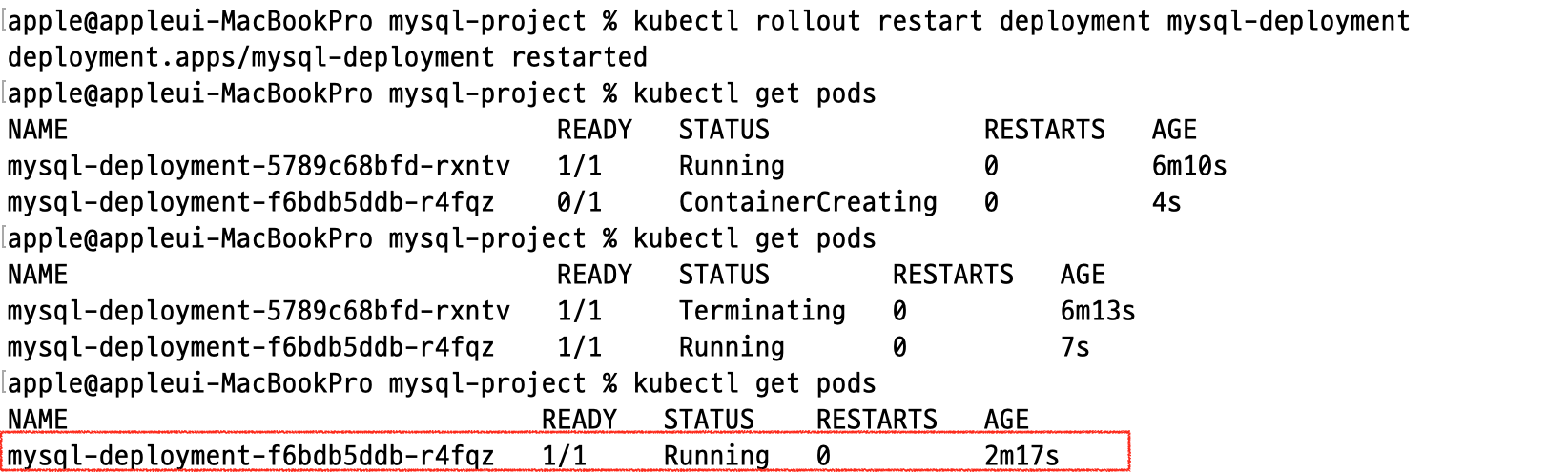
1. **매니페스트 파일 반영하기**

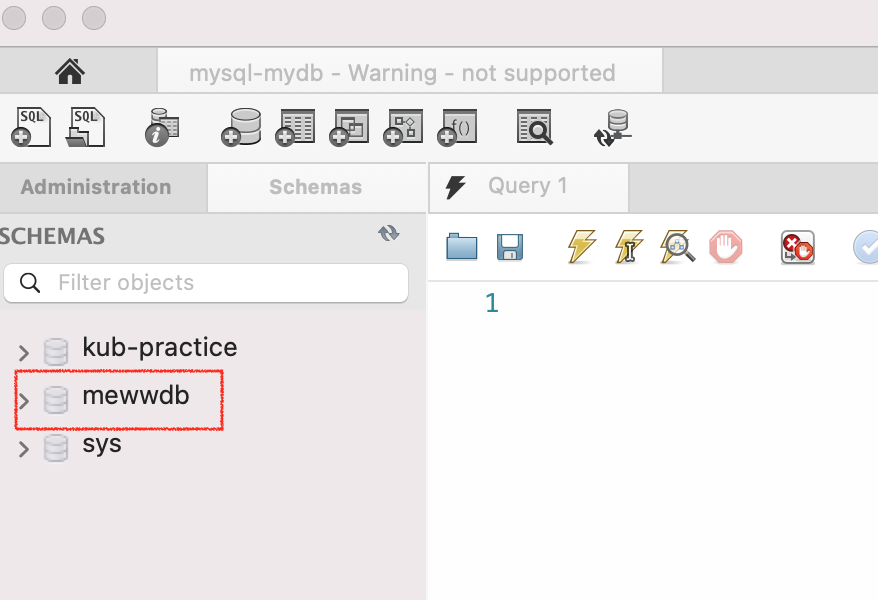
| $ kubectl apply -f mysql-pv.yaml  $ kubectl apply -f mysql-pvc.yaml  $ kubectl apply -f mysql-deployment.yaml |
| --- |

****

1. 새로운 데이터베이스 생성하기  
   
2. **디플로이먼트 재시작하기**

| $ kubectl rollout restart deployment mysql-deployment |
| --- |

****

1. **데이터베이스 다시 확인해보기**  
     
    디플로이먼트를 재시작하기 전에 생성했던 데이터베이스가 삭제되지 않고 그대로 잘 남아있는 걸 확인할 수 있다.  
   
2. **[예제] 백엔드(Spring Boot) 서버와 MySQL 연동하기**

### **✅ 백엔드(Spring Boot) 서버와 MySQL 연동하기**

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 Driver**를 선택해라.
2. **간단한 코드 작성**

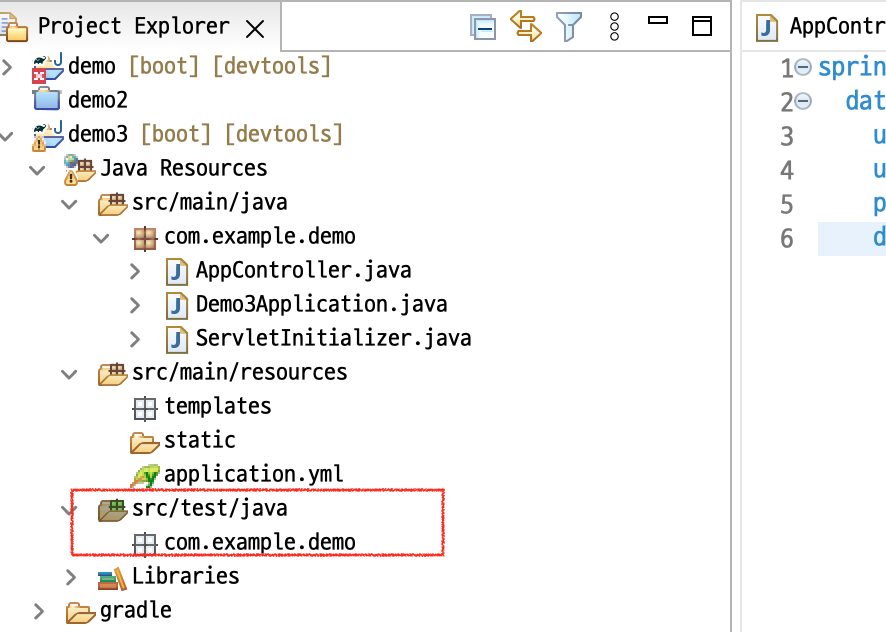
AppController

| @RestController  public class AppController {  @GetMapping("/")  public String home() {  return "Hello, World!";  }  } |
| --- |

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

application.yml

| spring:  datasource:  url: jdbc:mysql://${DB\_HOST}:${DB\_PORT}/${DB\_NAME}  username: ${DB\_USERNAME}  password: ${DB\_PASSWORD}  driver-class-name: com.mysql.cj.jdbc.Driver |
| --- |

1. **불필요한 테스트 코드 삭제  
   **
2. **Dockerfile 작성하기**

Dockerfile

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

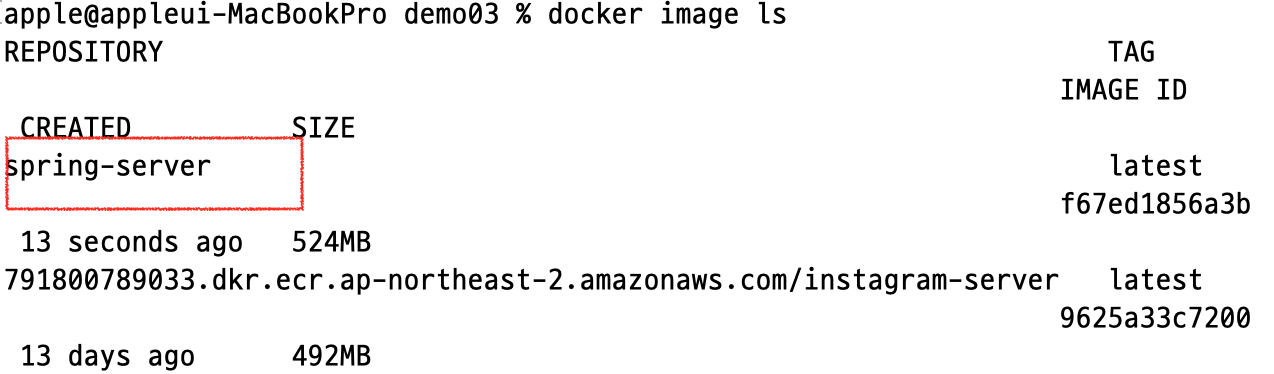
1. **Spring Boot 프로젝트 빌드하기**

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

****

1. **이미지 빌드하기**

| $ docker build -t spring-server . |
| --- |

****

1. **스프링 서버 띄우기 위한 매니페스트 파일 작성하기**

spring-deployment.yaml

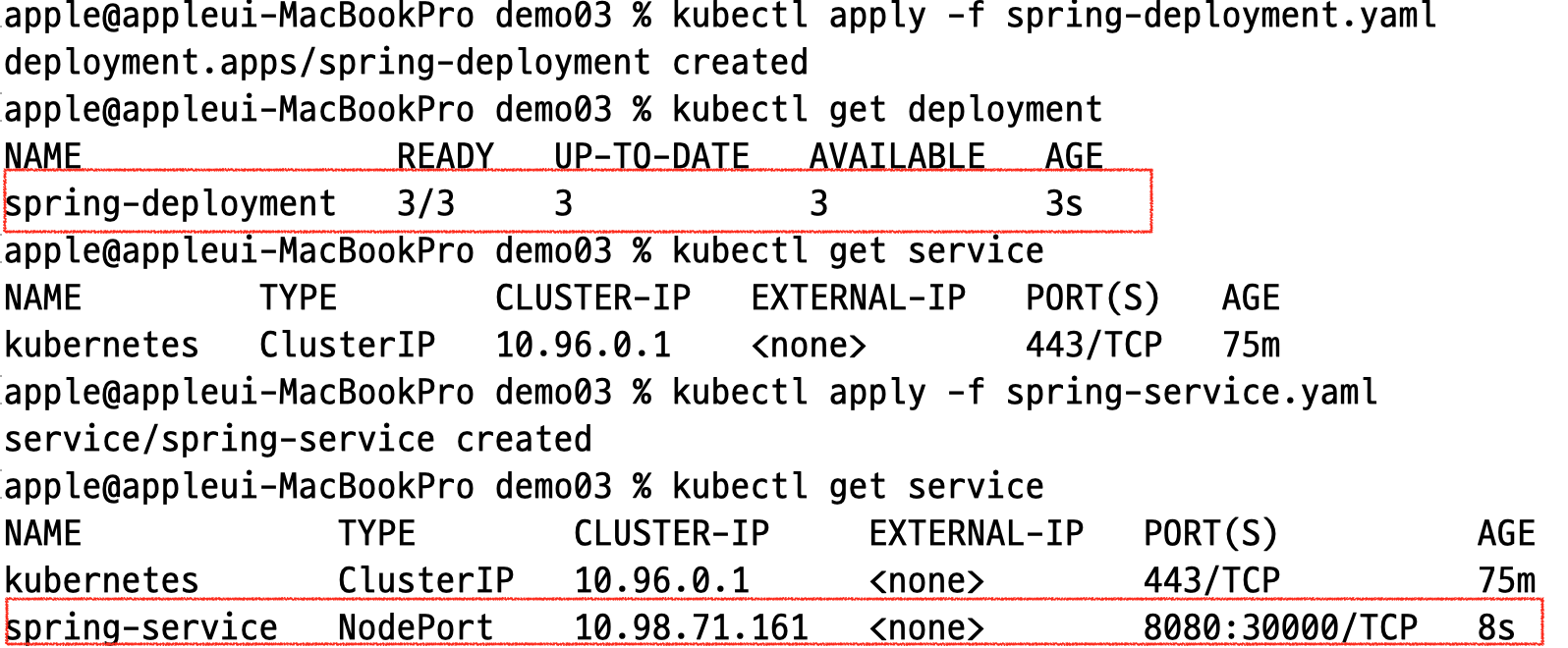
| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: backend-app  spec:  containers:  - name: spring-container # 컨테이너 이름  image: spring-server # 컨테이너를 생성할 때 사용할 이미지  imagePullPolicy: IfNotPresent # 로컬에서 이미지를 먼저 가져온다. 없으면 레지스트리에서 가져온다.  ports:  - containerPort: 8080 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  - name: DB\_HOST  value: mysql-service # Service의 name만 입력하면 다른 서비스와 통신할 수 있다.  \*\*\*\* - name: DB\_PORT  value: "3306" # 숫자값을 문자로 인식하게 만들기 위해 쌍따옴표 붙여야 한다.  - name: DB\_NAME  value: kub-practice  - name: DB\_USERNAME  value: root  - name: DB\_PASSWORD  value: password123 |
| --- |

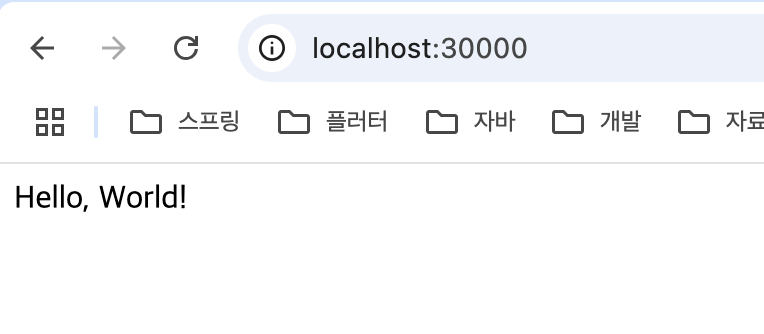
**spring-service.yaml**

| apiVersion: v1  kind: Service  # Service 기본 정보  metadata:  name: spring-service  # Service 세부 정보  spec:  type: NodePort # Service의 종류  selector:  app: backend-app # 실행되고 있는 파드 중 'app: backend-app'이라는 값을 가진 파드와 서비스를 연결  ports:  - protocol: TCP # 서비스에 접속하기 위한 프로토콜  port: 8080 # 쿠버네티스 내부에서 Service에 접속하기 위한 포트 번호 (Service  targetPort: 8080 # 매핑하기 위한 파드의 포트 번호  nodePort: 30000 # 외부에서 사용자들이 접근하게 될 포트 번호 |
| --- |

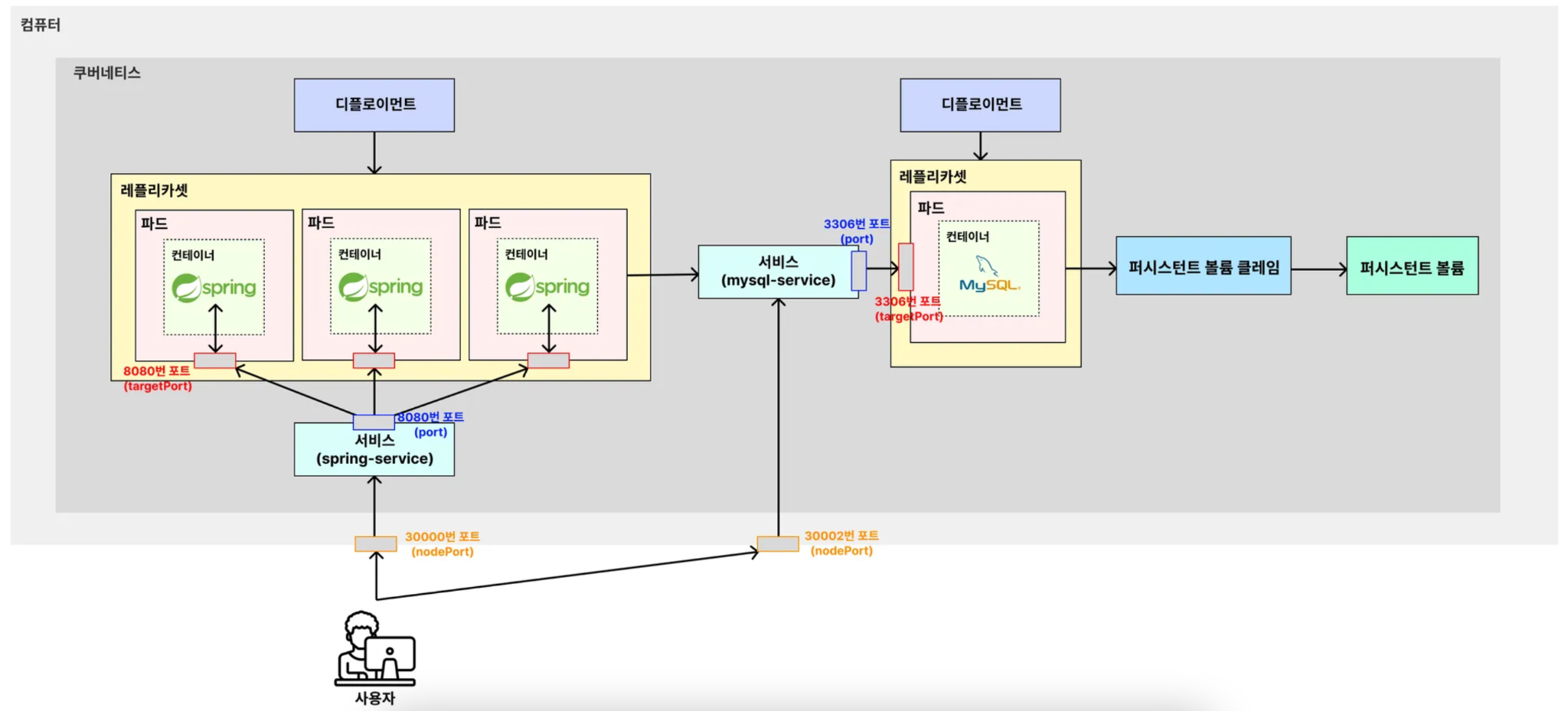
1. **매니페스트 파일 실행하기**

| $ kubectl apply -f spring-deployment.yaml  $ kubectl apply -f spring-service.yaml |
| --- |

****

1. **MySQL과 연결된 백엔드 서버 잘 작동하는 지 확인하기  
     
   **

### **✅ 그림으로 이해하기**

****

1. **만약 MySQL과 연결이 제대로 되지 않았다면 어떻게 될까?**

### **✅ 만약 MySQL과 연결이 제대로 되지 않았다면 어떻게 될까?**

1. **매니페스트 파일 수정하기**

### **DB 정보를 일부러 틀리게 바꿔보자. spring-deployment.yaml**

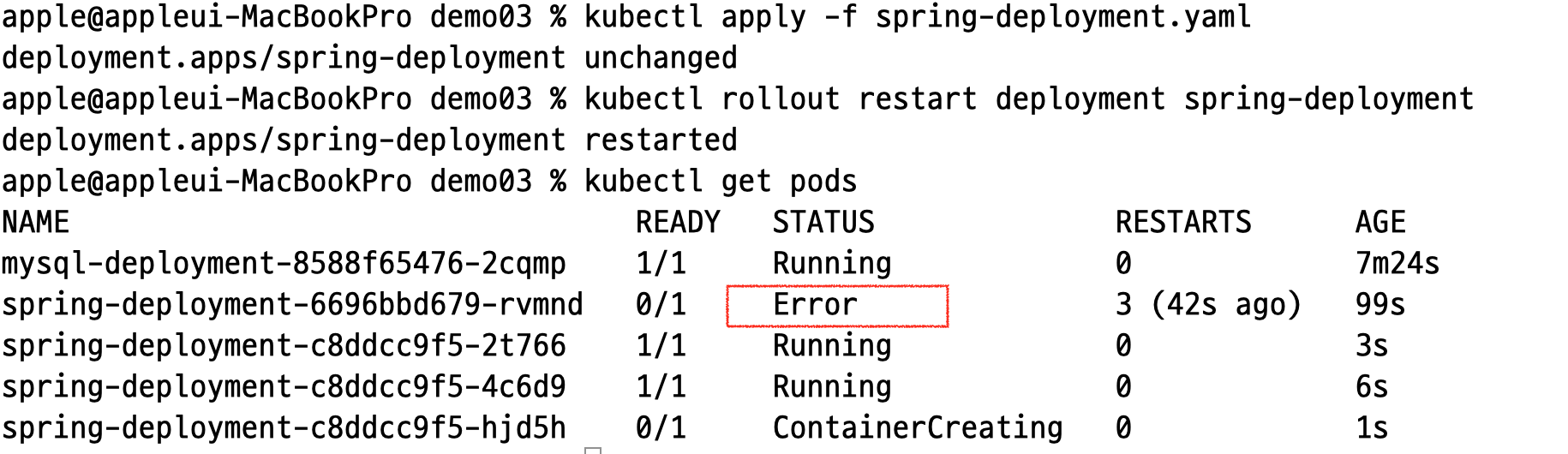
| apiVersion: apps/v1 kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: backend-app  spec:  containers:  - name: spring-container # 컨테이너 이름  image: spring-server # 컨테이너를 생성할 때 사용할 이미지  imagePullPolicy: IfNotPresent # 로컬에서 이미지를 먼저 가져온다. 없으면 레지스트리에서 가져온다.  ports:  - containerPort: 8080 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  - name: DB\_HOST  value: mysql-service # Service의 name만 입력하면 다른 서비스와 통신할 수 있다.  - name: DB\_PORT  value: "3306" # 숫자값을 문자로 인식하게 만들기 위해 쌍따옴표 붙여야 한다.  - name: DB\_NAME  value: kub-practice  - name: DB\_USERNAME  value: root  - name: DB\_PASSWORD  value: wrong-password |
| --- |

1. **디플로이먼트 적용 및 재시작**

| **$ kubectl apply -f spring-deployment.yaml**  **$ kubectl rollout restart deployment spring-deployment** |
| --- |

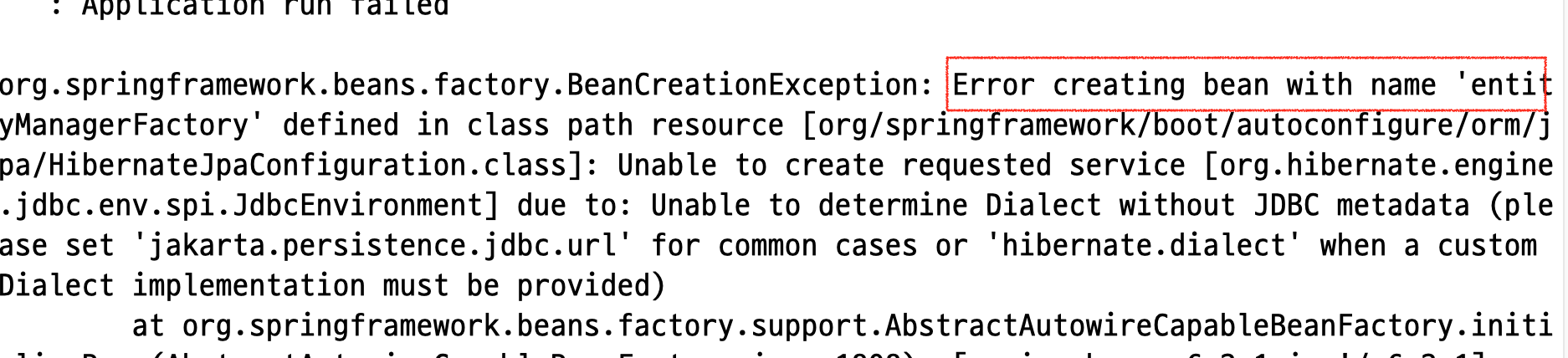
1. **잘 작동하는 지 확인**

| $ kubectl get pods |
| --- |

****

**[에러 메시지 확인하기]** DB 정보가 잘못돼서 연결이 제대로 안 되면 아래와 같은 에러 메시지가 발생하면서 파드가 실행되지 않는다.

| $ kubectl logs [파드명] |
| --- |



1. **다시 DB 정보를 올바르게 바꾸고 실행시키기**

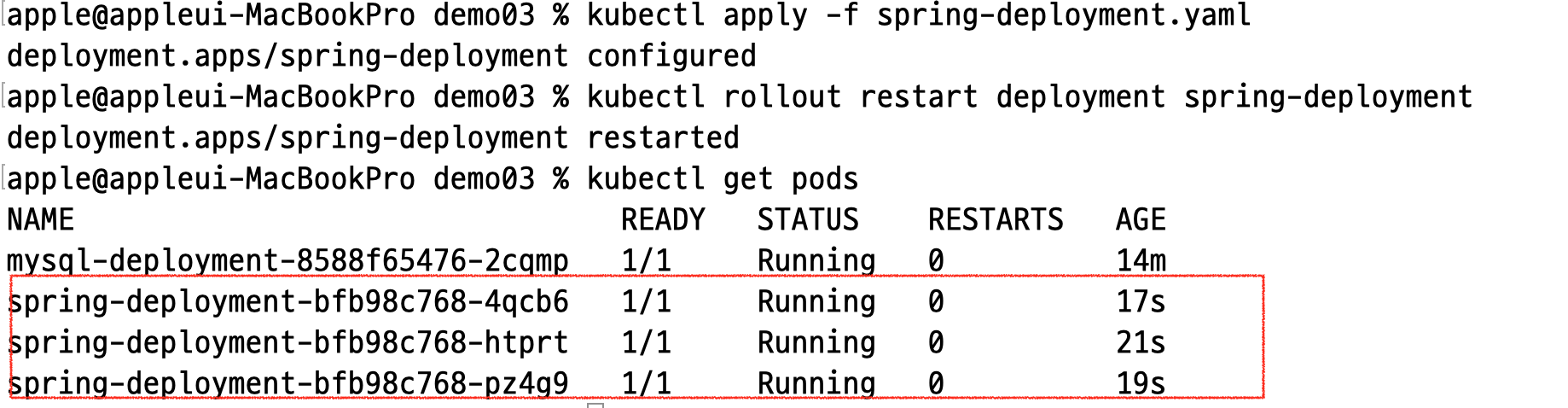
spring-deployment.yaml

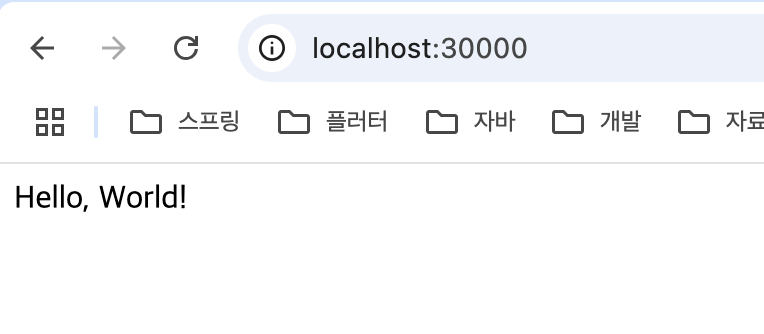
| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: backend-app  spec:  containers:  - name: spring-container # 컨테이너 이름  image: spring-server # 컨테이너를 생성할 때 사용할 이미지  imagePullPolicy: IfNotPresent # 로컬에서 이미지를 먼저 가져온다. 없으면 레지스트리에서 가져온다.  ports:  - containerPort: 8080 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  - name: DB\_HOST  value: mysql-service # Service의 name만 입력하면 다른 서비스와 통신할 수 있다.  - name: DB\_PORT  value: "3306" # 숫자값을 문자로 인식하게 만들기 위해 쌍따옴표 붙여야 한다.  - name: DB\_NAME  value: kub-practice  - name: DB\_USERNAME  value: root  - name: DB\_PASSWORD  value: **password123** |
| --- |

| $ kubectl apply -f spring-deployment.yaml  $ kubectl rollout restart deployment spring-deployment |
| --- |

**정상적으로 작동되는 걸 확인할 수 있다.**

| $ kubectl get pods |
| --- |

****

****

1. **컨피그맵(ConfigMap), 시크릿(Secret)을 활용해 환경변수 분리하기**
2. **ConfigMap 매니페스트 파일 생성하기**

spring-config.yaml

| apiVersion: v1  kind: ConfigMap  # ConfigMap 기본 정보  metadata:  name: spring-config # ConfigMap 이름  # Key, Value 형식으로 설정값 저장  data:  db-host: mysql-service  db-port: "3306"  db-name: kub-practice |
| --- |

spring-deployment.yaml

| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: backend-app  spec:  containers:  - name: spring-container # 컨테이너 이름  image: spring-server # 컨테이너를 생성할 때 사용할 이미지  imagePullPolicy: IfNotPresent # 로컬에서 이미지를 먼저 가져온다. 없으면 레지스트리에서 가져온다.  ports:  - containerPort: 8080 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  **- name: DB\_HOST**  **valueFrom:**  **configMapKeyRef:**  **name: spring-config**  **key: db-host**  **- name: DB\_PORT**  **valueFrom:**  **configMapKeyRef:**  **name: spring-config**  **key: db-port**  **- name: DB\_NAME**  **valueFrom:**  **configMapKeyRef:**  **name: spring-config**  **key: db-name**  - name: DB\_USERNAME  value: root  - name: DB\_PASSWORD  value: password123 |
| --- |

1. **Secret 매니페스트 파일 만들기**

spring-secret.yaml

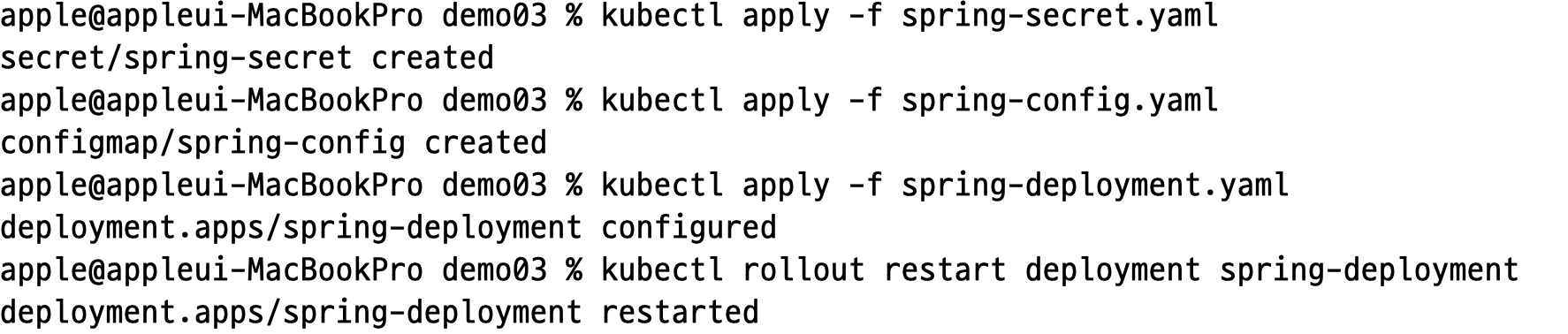
| apiVersion: v1  kind: Secret  # Secret 기본 정보  metadata:  name: spring-secret # Secret 이름  # Key, Value 형식으로 값 저장  stringData:  db-username: root  db-password: password123 |
| --- |

spring-deployment.yaml

| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: backend-app  spec:  containers:  - name: spring-container # 컨테이너 이름  image: spring-server # 컨테이너를 생성할 때 사용할 이미지  imagePullPolicy: IfNotPresent # 로컬에서 이미지를 먼저 가져온다. 없으면 레지스트리에서 가져온다.  ports:  - containerPort: 8080 # 컨테이너에서 사용하는 포트를 명시적으로 표현  env:  - name: DB\_HOST  valueFrom:  configMapKeyRef:  name: spring-config  key: db-host  - name: DB\_PORT  valueFrom:  configMapKeyRef:  name: spring-config  key: db-port  - name: DB\_NAME  valueFrom:  configMapKeyRef:  name: spring-config  key: db-name  **- name: DB\_USERNAME**  **valueFrom:**  **secretKeyRef:**  **name: spring-secret**  **key: db-username**  **- name: DB\_PASSWORD**  **valueFrom:**  **secretKeyRef:**  **name: spring-secret**  **key: db-password** |
| --- |

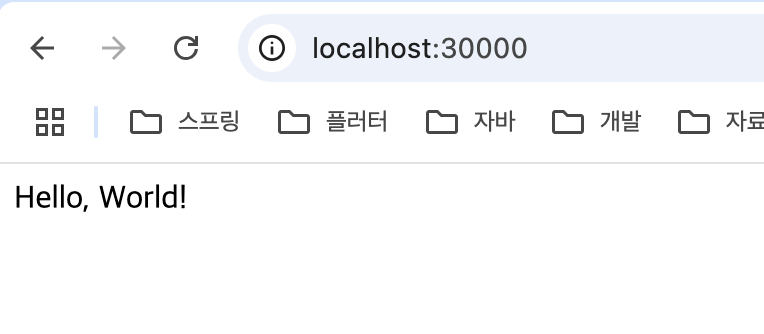
1. 매니페스트 파일 반영하기

| $ kubectl apply -f spring-secret.yaml  $ kubectl apply -f spring-config.yaml  $ kubectl apply -f spring-deployment.yaml  $ kubectl rollout restart deployment spring-deployment |
| --- |



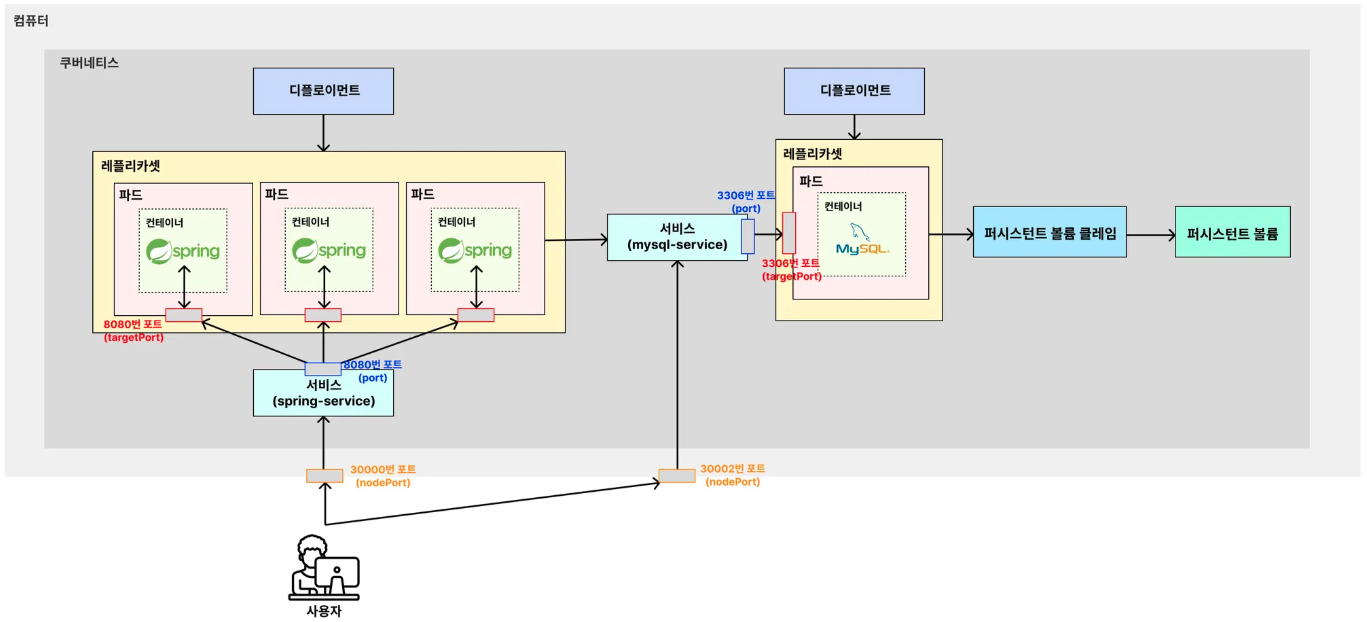
1. 잘 반영 됐는지 확인하기



****

1. **보안을 위해 외부에서 MySQL 접근하지 못하도록 막기**

### **✅ 기존 구성의 보안적인 문제점**

****

지금까지 만든 구조를 살펴보면 위와 같다. 위의 구조에서 MySQL을 중점적으로 살펴보면 보안에 취약한 점이 있다. **30002**번 포트로 MySQL에 직접적으로 접근할 수 있게끔 보안이 설정되어 있다는 점이다. **Service**의 **NodePort**를 활용해 **30002**번 포트를 외부로 오픈해서 MySQL에 아무나 접근할 수 있게 만들었다.

### **✅ Service 종류 복습**

* **NodePort** : 쿠버네티스 내부에서 해당 서비스에 접속하기 위한 포트를 열고 외부에서 접속 가능하도록 한다.
* **ClusterIP** : 쿠버네티스 내부에서만 통신할 수 있는 IP 주소를 부여. 외부에서는 요청할 수 없다.
* **LoadBalancer** : 외부의 로드밸런서(AWS의 로드밸런서 등)를 활용해 외부에서 접속할 수 있도록 연결한다.

### **✅ 보안적인 문제점 해결**

보안적인 문제점 해결을 위해 **Service**의 종류 중 **NodePort**를 사용하지 않고 **ClusterIP**를 활용해야 한다. **ClusterIP**를 활용함으로써 외부에서 아무나 MySQL에 접근하지 못하게 막아야 한다.

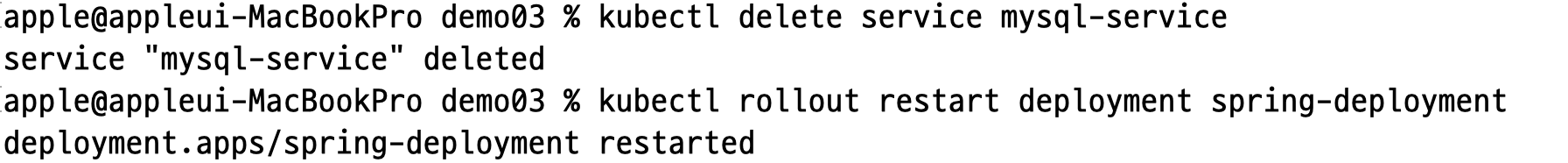
1. **기존 Service 매니페스트 파일 수정하기**

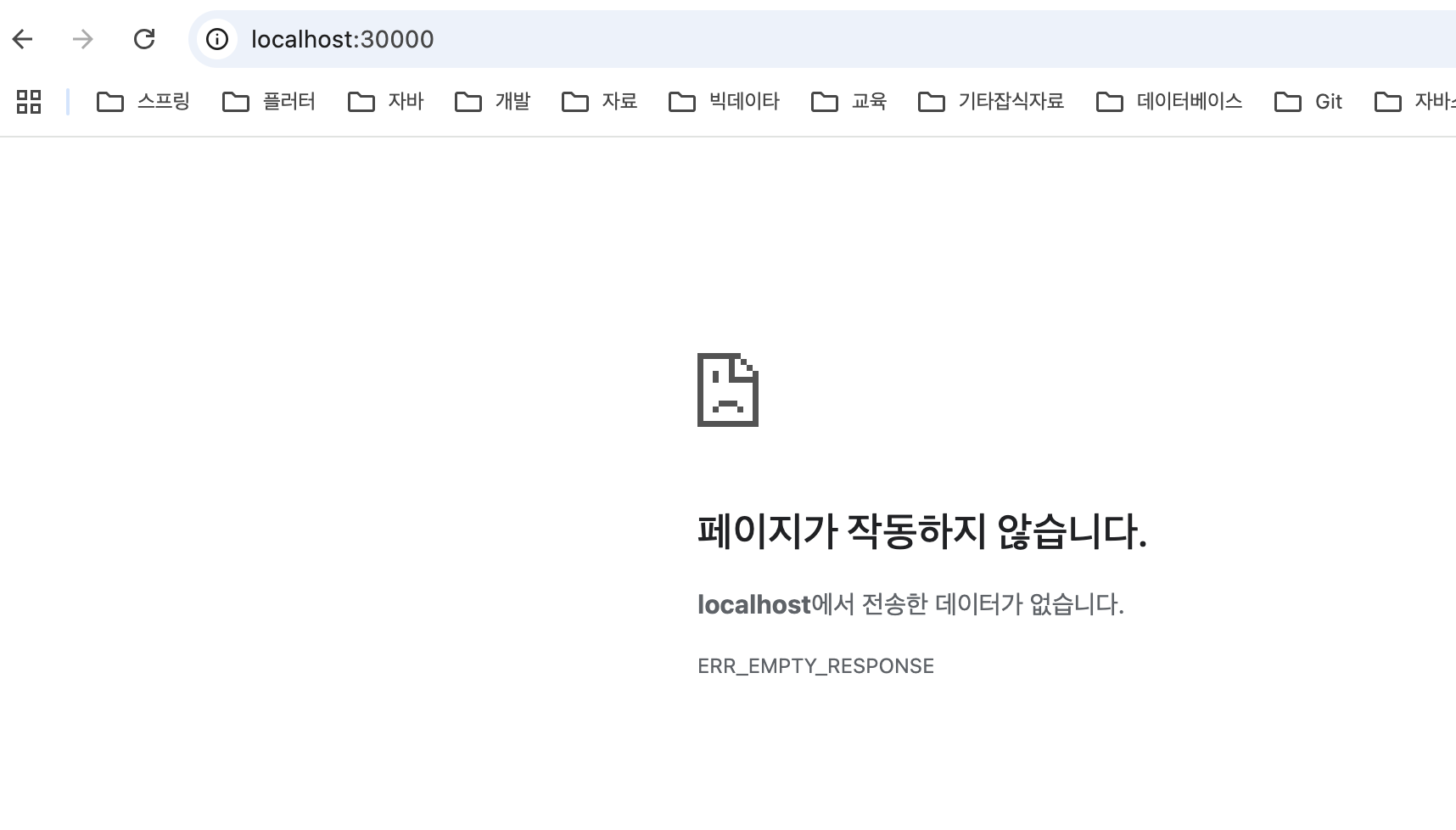
mysql-service.yaml

| apiVersion: v1  kind: Service  # Service 기본 정보  metadata:  name: mysql-service # Service 이름  # Service 세부 정보  spec:  type: **ClusterIP** # Service의 종류  selector:  app: mysql-db # 실행되고 있는 파드 중 'app: mysql-db'이라는 값을 가진 파드와 서비스를 연결  ports:  - protocol: TCP # 서비스에 접속하기 위한 프로토콜  port: 3306 # 쿠버네티스 내부에서 Service에 접속하기 위한 포트 번호  targetPort: 3306 # 매핑하기 위한 파드의 포트 번호  ~~nodePort: 30002 # 외부에서 사용자들이 접근하게 될 포트 번호~~ |
| --- |

1. **기존 Service 중단하면 작동 안 하는 지 먼저 확인하기**

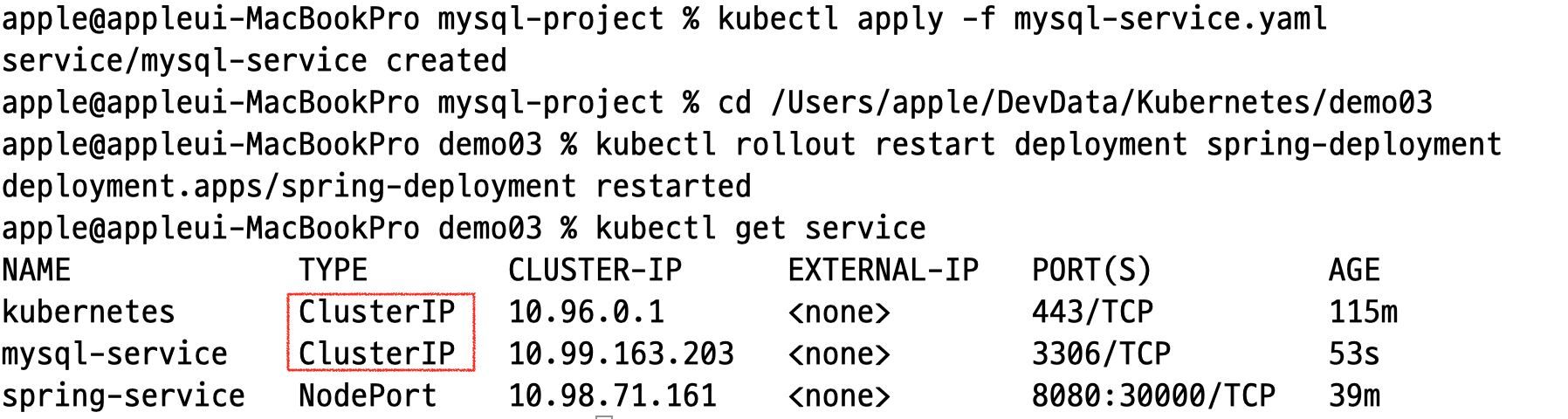
| $ kubectl delete service mysql-service  $ kubectl rollout restart deployment spring-deployment |
| --- |

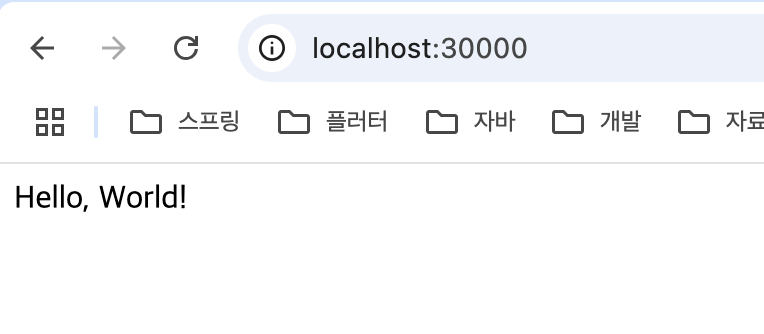
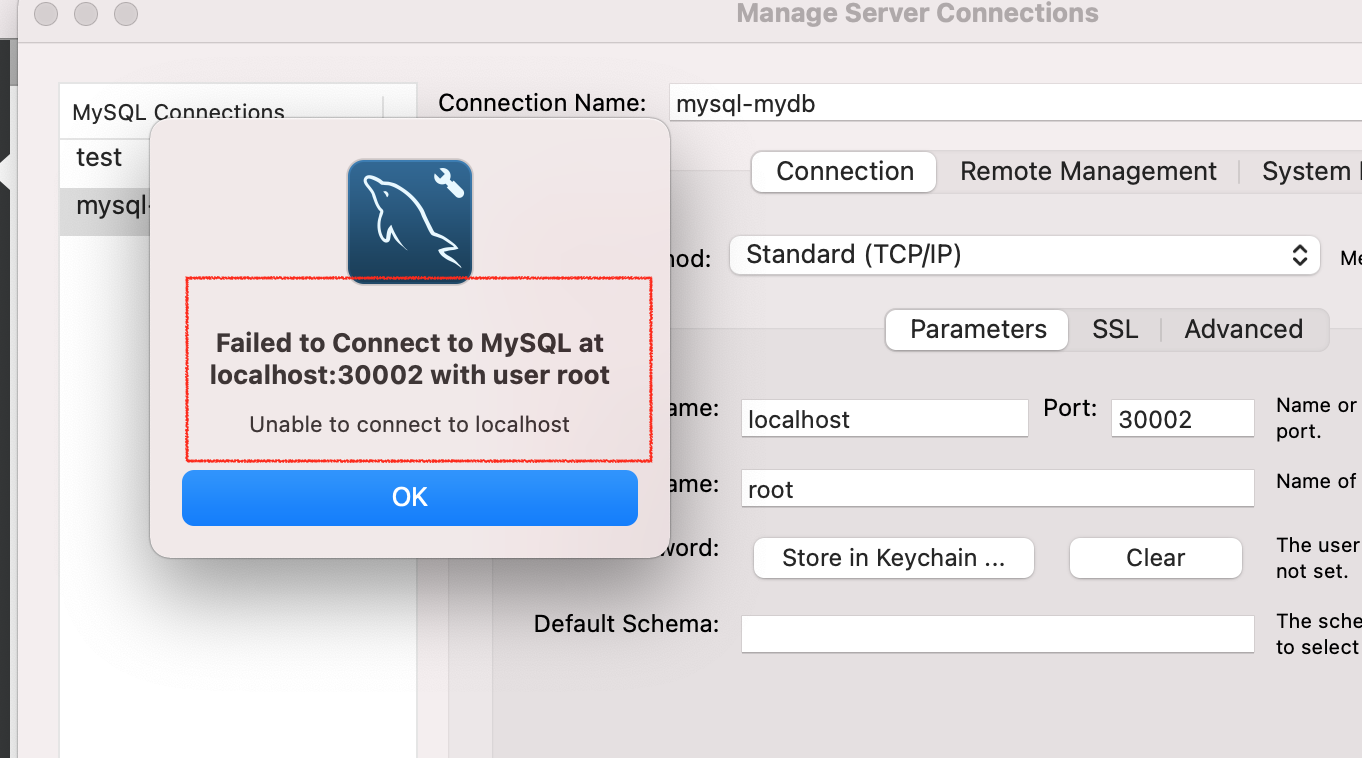




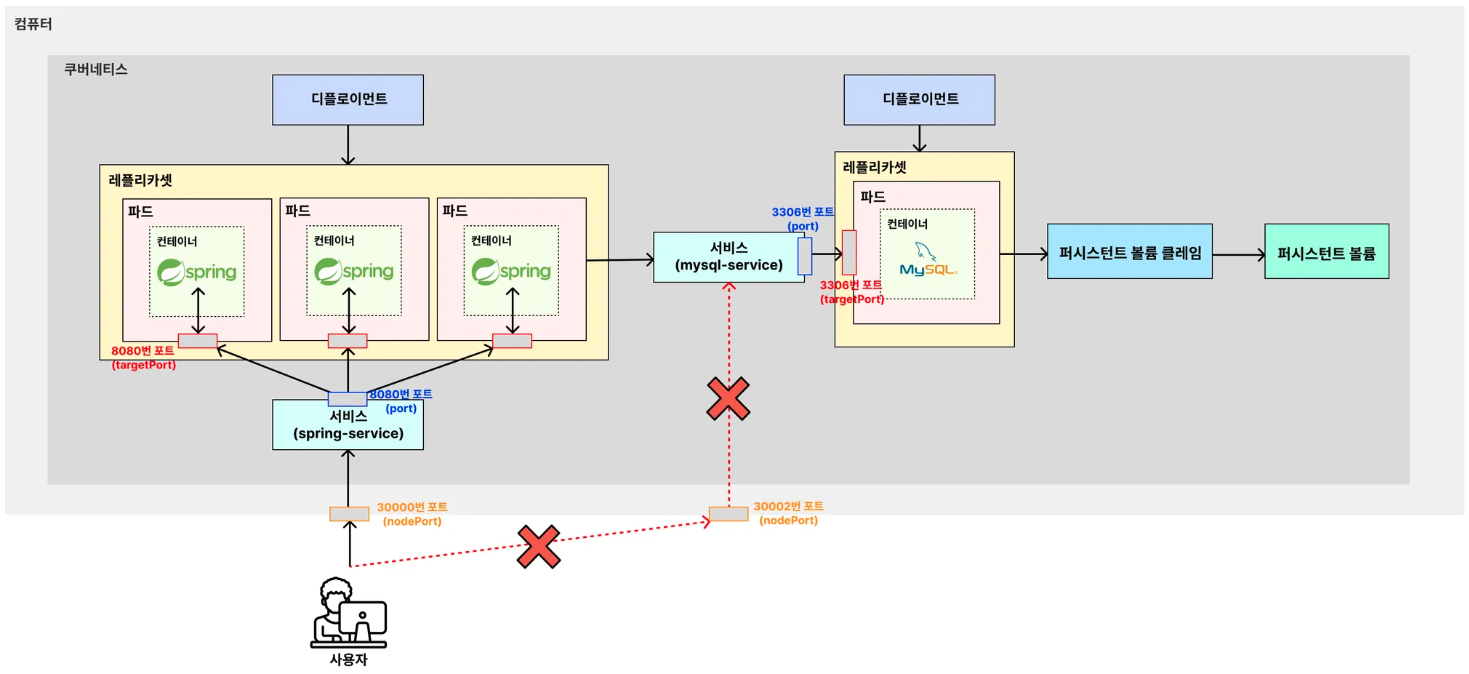
1. **Service 작동시키기**

| $ kubectl apply -f mysql-service.yaml  $ kubectl rollout restart deployment spring-deployment |
| --- |

****

1. **Spring Boot 서버와 잘 연결됐는 지 확인**  
     
    ****
2. **원래 접속하던 DB에 접속이 안 되는 지 확인**  
   

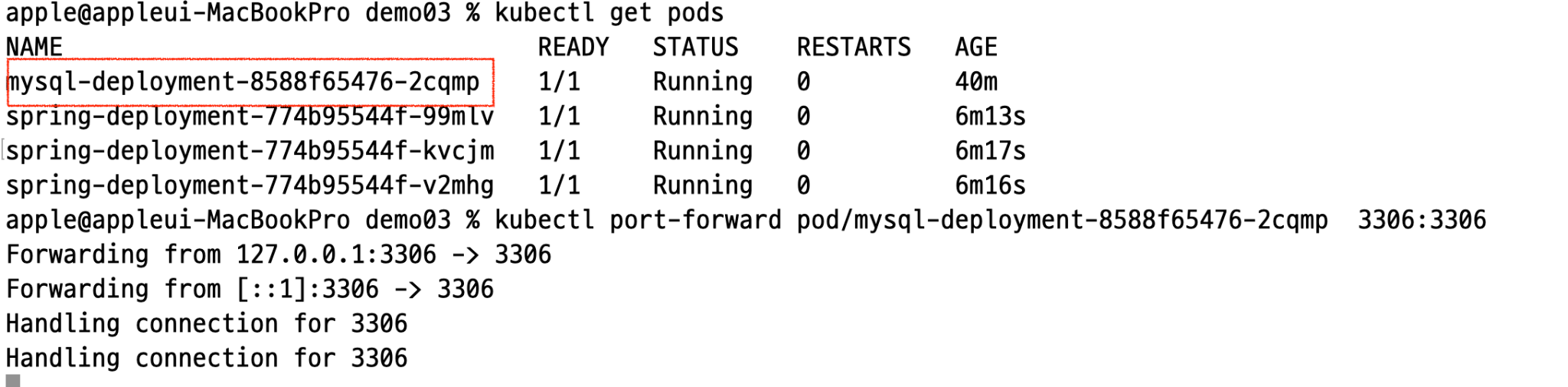
### **✅ 그림으로 이해하기**

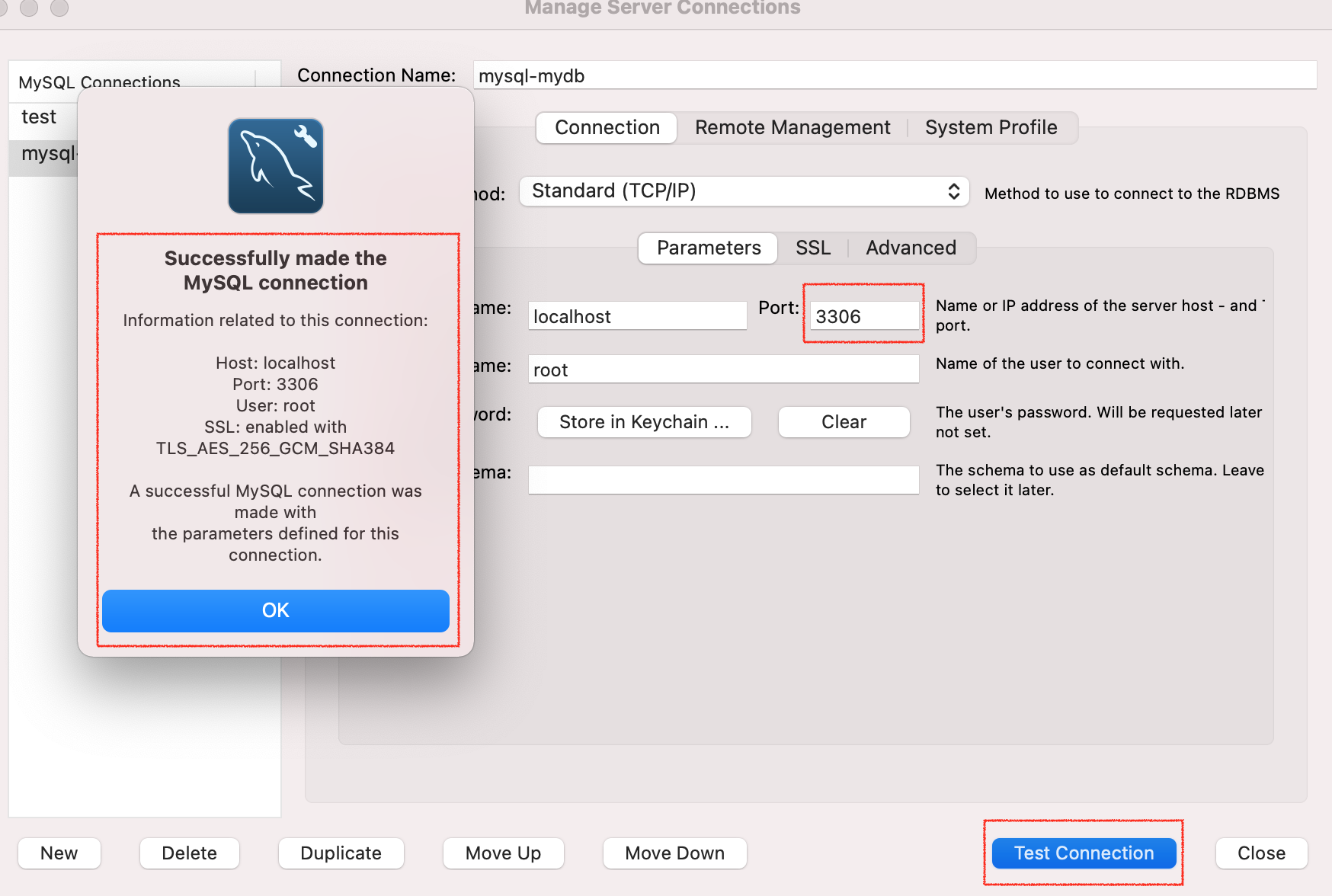


### **✅ DB를 관리하기 위해 접속해야 할 때는?**

쿠버네티스의 포트 포워딩을 활용해서 접속하면 된다. 아래 포트 포워딩 명령어를 사용하면 내 로컬 컴퓨터에서만 해당 파드와 연결을 허용시킬 수 있게 된다.

| $ kubectl port-forward pod/[MySQL 파드명] 3306:3306 |
| --- |



****