# **04.컨피그맵(ConfigMap), 시크릿(Secret)을 활용해 환경변수 관리하기**

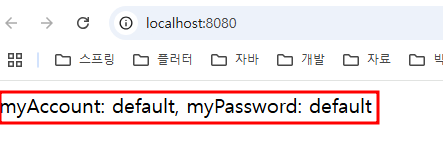
1. **[예제] 백엔드(Spring Boot) 서버에 환경변수 등록해 사용하기**

### **✅ 백엔드(Spring Boot) 서버에 환경변수 등록해 사용하기**

1. **Spring Boot 프로젝트 셋팅(demo2)** [start.spring.io](https://start.spring.io/) 의존성 추가 web, devtools
2. **간단한 코드 작성**

AppController

| @RestController  public class AppController {  @Value("${MY\_ACCOUNT:default}")  private String myAccount;    @Value("${MY\_PASSWORD:default}")  private String myPassword;  @GetMapping("/")  public String home() {  return "myAccount: " + myAccount + ", myPassword: " + myPassword;  }  } |
| --- |

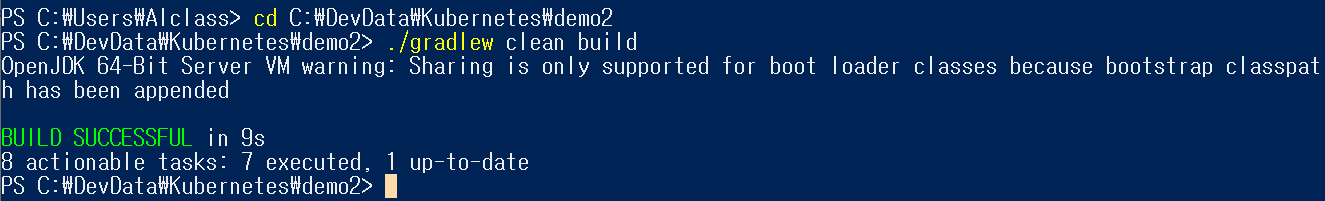
1. **프로젝트 실행시켜보기  
   **
2. **Dockerfile 작성하기**

**Dockerfile**

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

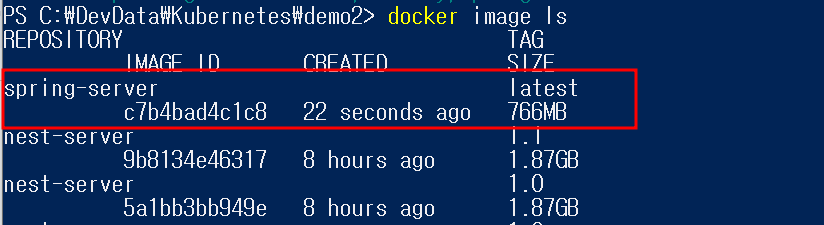
1. **Spring Boot 프로젝트 빌드하기**

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



1. **Dockerfile을 바탕으로 이미지 빌드하기**

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

****

1. **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:** # 환경변수 등록  **- name: MY\_ACCOUNT**  **value: 본인ID**  **- name: MY\_PASSWORD**  **value: pwd1234** |
| --- |

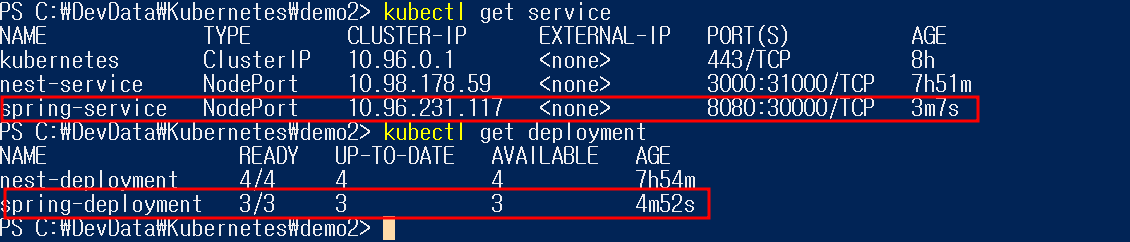
**spring-service.yaml**

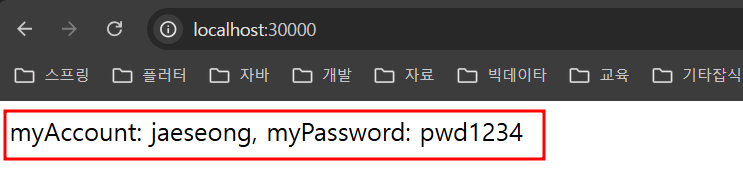
| apiVersion: v1  kind: Service  # Service 기본 정보  metadata:  name: spring-service    # Service 세부 정보  spec:  type: NodePort # Service의 종류  selector:  app: backend-app # 실행되고 있는 파드 중 'app: backend-app'이라는 값을 가진 파드와 서비스를 연결  ports:  - protocol: TCP # 서비스에 접속하기 위한 프로토콜  port: 8080 # 쿠버네티스 내부에서 Service에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

1. **매니페스트 기반으로 실행시키기**

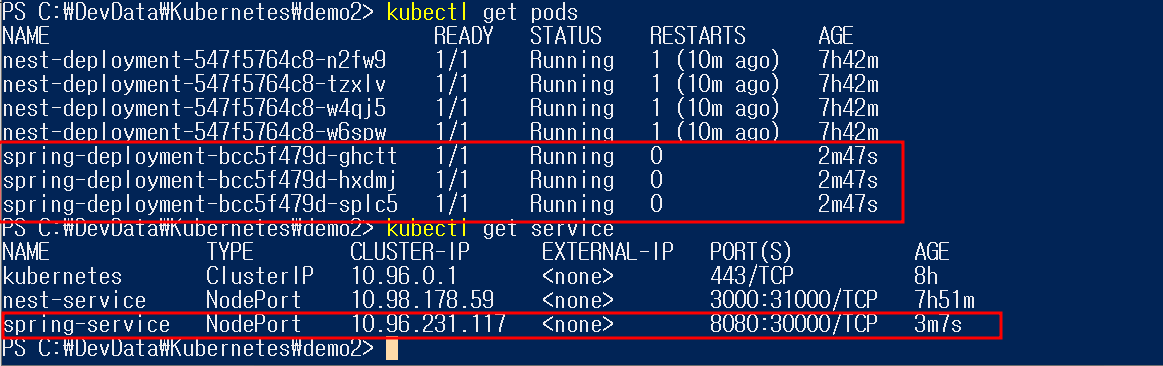
| $ kubectl apply -f spring-deployment.yaml  $ kubectl apply -f spring-service.yaml |
| --- |

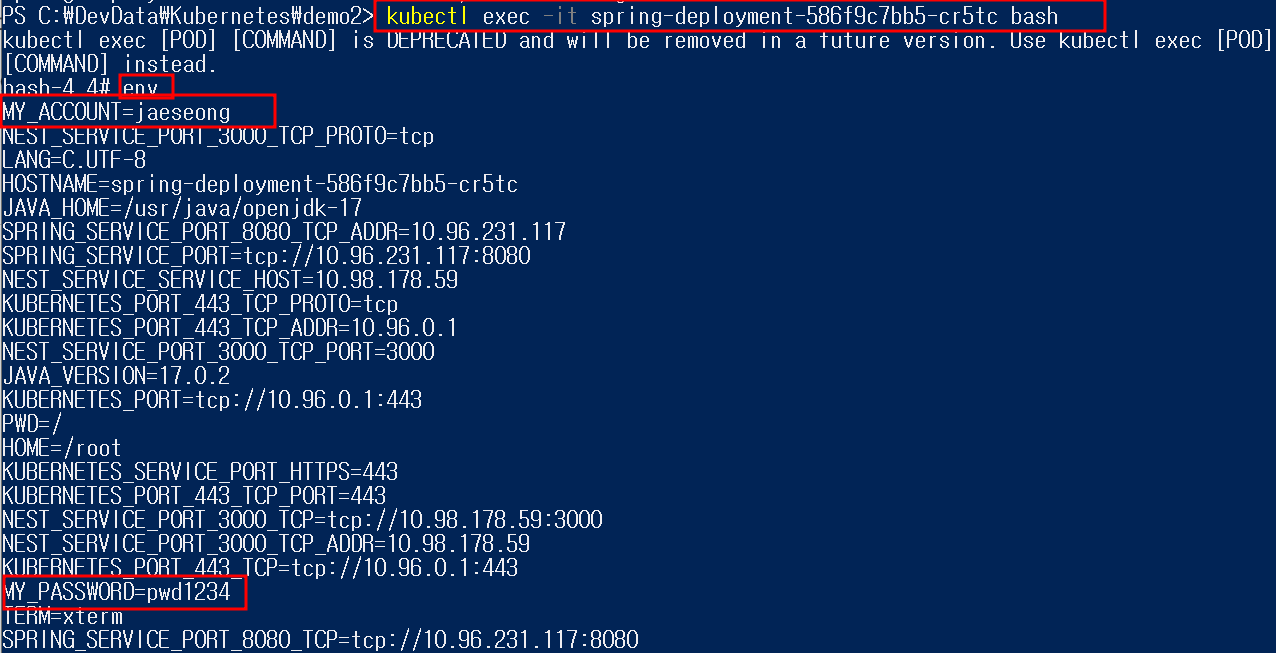
확인하기



1. **환경변수가 잘 적용 됐는지 확인해보기**
2. **파드 내부로 접속해서 확인해보기**

| $ kubectl get pods # 파드명 확인하기  $ kubectl exec -it [파드명] -- bash # 파드 내부로 접속하기  $ env # 환경변수 조회 |
| --- |





1. **컨피그맵(ConfigMap)을 활용해 환경변수 분리하기**

### **✅ 컨피그맵(ConfigMap)이란?**

Spring Boot에서는 설정값을 *application.yml*으로 분리해서 관리한다. Nest.js에서도 설정값을 **.env**으로 분리해서 관리한다. 별도의 파일로 분리를 해서 관리함으로써 유지보수가 편리해지고 개발, 테스트, 프로덕션과 같은 환경 분리가 편해진다.

쿠버네티스에서는 파드(Pod), 디플로이먼트(Deployment), 서비스(Service)가 각각의 역할을 가지고 있는 것처럼 환경 변수를 관리하는 역할을 가진 오브젝트가 따로 존재한다. 그게 바로 **컨피그맵(ConfigMap)**이다.

### **✅ 디플로이먼트(Deployment)에 환경 변수의 정보를 같이 작성했을 때의 단점**

**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:** # 환경변수 등록  **- name: MY\_ACCOUNT** # Key 값  **value: 본인ID** # Value 값  **- name: MY\_PASSWORD**  **value: pwd1234** |
| --- |

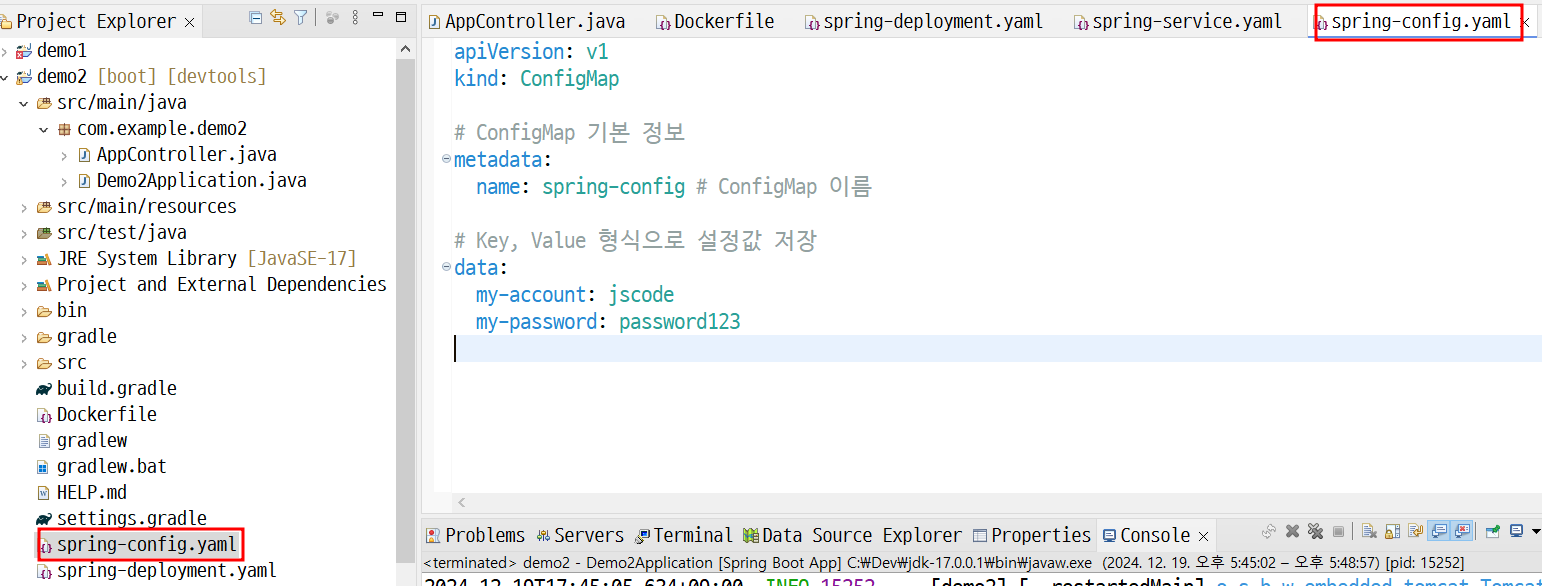
위 매니페스트 파일을 보면 디플로이먼트(Deployment)에 대한 내용과 환경 변수에 관련된 내용을 같이 작성했다. 이렇게 환경 변수를 디플로이먼트(Deployment) 내부에 작성하면 다른 환경(개발, 테스트, 프로덕션 등)에서 서버를 실행할 때 유연하게 설정 값을 변경하기 어려워진다.

### **✅ 컨피그맵(ConfigMap)을 활용해 환경변수 분리하기**

1. **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:  my-account: kmdadoo1  my-password: password123 |
| --- |

****

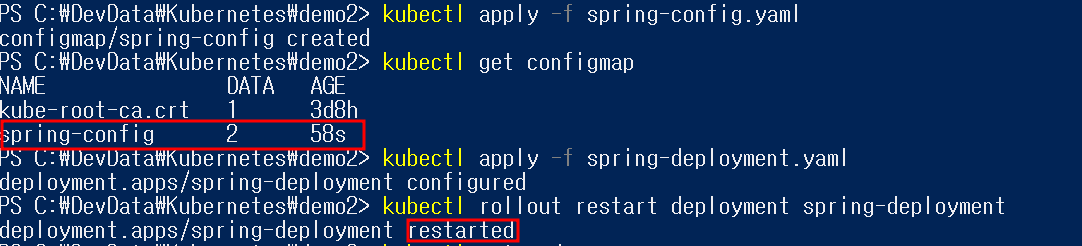
* + **참고)** 컨피그맵이 잘 적용됐는 지 확인하기 위해 **my-account, my-password** 값 변경

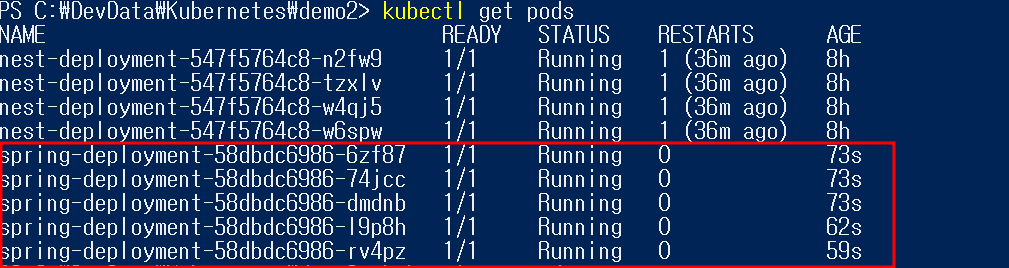
1. **Deployment 매니페스트 파일 수정하기**

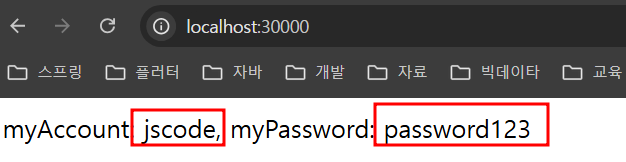
| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 5 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  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: MY\_ACCOUNT**  **valueFrom:**  **configMapKeyRef:**  **name: spring-config** # ConfigMap의 이름  **key: my-account** # ConfigMap에 설정되어 있는 Key값  **- name: MY\_PASSWORD**  **valueFrom:**  **configMapKeyRef:**  **name: spring-config**  **key: my-password** |
| --- |

1. **매니페스트 파일 반영하기**

| $ kubectl apply -f spring-config.yaml  $ kubectl apply -f spring-deployment.yaml  # kubectl rollout restart deployment [디플로이먼트명]  $ kubectl rollout restart deployment spring-deployment # Deployment 재시작 |
| --- |

****

****

1. **잘 반영 됐는지 확인하기  
   **
2. **시크릿(Secret)을 활용해 ‘민감한 값’을 환경 변수로 분리하기**

### **✅ 시크릿(Secret)이란?**

시크릿(Secret)은 컨피그맵(ConfigMap)과 비슷하게 환경 변수를 분리해서 관리하는 오브젝트이다. 차이점은 시크릿(Secret)은 비밀번호와 같이 보안적으로 중요한 값을 관리하기 위한 오브젝트이다.

### **✅ 시크릿(Secret)을 활용해 ‘민감한 값’을 따로 분리하기**

1. **기존 매니페스트 파일 살펴보기**

spring-config.yaml

| apiVersion: v1  kind: ConfigMap  # ConfigMap 기본 정보  metadata:  name: spring-config # ConfigMap 이름  # Key, Value 형식으로 설정값 저장  data:  my-account: thejoeun  **my-password: password123** |
| --- |

위 매니페스트 파일에서 **my\_password**의 값이 보안적으로 중요한 값이라고 가정해보자. 그러면 **my\_password**의 값은 컨피그맵(ConfigMap)이 아닌 시크릿(Secret)으로 관리해야 한다.

1. **기존 매니페스트 파일 수정하기 / 새로운 매니페스트 파일 생성하기**

수정 (spring-config.yaml)

| apiVersion: v1  kind: Secret  # ConfigMap 기본 정보  metadata:  name: spring-config # ConfigMap 이름  # Key, Value 형식으로 설정값 저장  data:  my-account: jscode  ~~my-password: password123~~ |
| --- |

생성 (spring-secret.yaml)

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

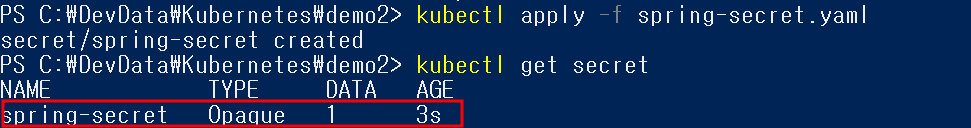
* + **참고)** 시크릿이 잘 적용 됐는지 확인하기 위해 **my-password** 값 변경

수정 (spring-deployment.yaml)

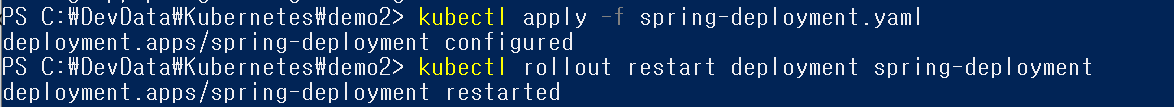
| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 5 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  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: MY\_ACCOUNT  valueFrom:  configMapKeyRef:  name: spring-config # ConfigMap의 이름  key: my-account # ConfigMap에 설정되어 있는 Key값  - name: MY\_PASSWORD  valueFrom:  **secretKeyRef:**  **name: spring-secret**  **key: my-password** |
| --- |

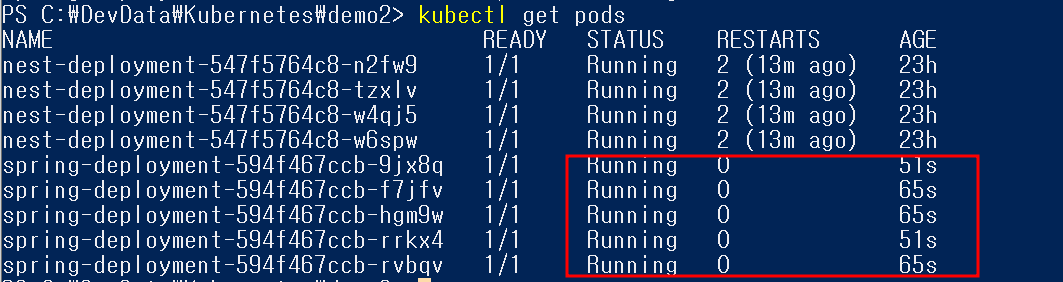
1. **매니페스트 파일 반영하기**

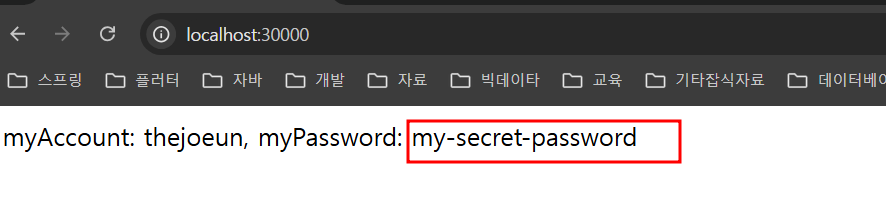
| $ kubectl apply -f spring-secret.yaml  $ kubectl apply -f spring-config.yaml  $ kubectl apply -f spring-deployment.yaml  $ kubectl rollout restart deployment spring-deployment |
| --- |

****

****

****

****

1. 잘 반영 됐는지 확인하기 ****