# **06.AWS EC2에서 쿠버네티스를 활용해 서버 배포하기**

1. **EC2에서 쿠버네티스를 활용했을 때 발생하는 AWS 예상 비용**

### **✅ EC2**

* EC2 인스턴스 (t4g.small) : 시간당 USD 0.0208 (24시간당 약 700원)  
    
   (비용이 걱정되시는 분들은 학습이 끝나자마자 인스턴스를 종료하시길 권장드립니다.)
* 데이터 전송 비용 : 1 GB당 0.1368 USD (1GB당 약 200원)  
    
   (실습 과정 동안 1GB 이하의 데이터만 전송합니다.)
* Public IPv4 비용 : 시간당 0.005 USD (24시간당 약 200원)

### **✅ RDS**

* RDS 인스턴스 (t4g.micro) : 시간당 USD 0.025 (24시간당 약 800원)  
    
   (프리티어일 경우 월 750시간까지 무료)
* 스토리지 비용 : GB-월당 0.131 USD (20GB-24시간당 약 200원)  
    
   (프리티어일 경우 20GB까지 무료)
* Public IPv4 비용 : 시간당 0.005 USD (24시간당 약 200원)

### **✅ ECR**

* 스토리지 비용 : GB-월당 USD 0.10 (1GB-24시간당 약 10원)  
    
   (실습 과정 동안 1GB 이하의 데이터만 저장합니다.)
* 데이터 전송 비용 : GB-월당 USD 0.126 (1GB-24시간당 약 6원)  
    
   (실습 과정 동안 1GB 이하의 데이터만 전송합니다.)

1. **k3s란?**

### **✅ EC2에서 쿠버네티스(k8s)를 쓰는 게 학습에 도움이 안 되는 이유**

쿠버네티스의 학습을 가로막는 요소에는 크게 2가지가 있다.

1. **복잡한 설치 과정**  
     
   쿠버네티스를 입문하는 입장이거나 Devops 엔지니어가 되려는 사람이 아니라면 쿠버네티스를 직접 설치할 일은 극히 드물다. 심지어 Devops 엔지니어라고 할 지라도 최근에는 쿠버네티스를 직접 설치하는 일은 드물다. 왜냐하면 AWS 자체에서 쿠버네티스 서비스(EKS)를 제공하고 있기 때문이다.
2. **EC2 인스턴스를 3대 만들어야 한다는 비용적인 부담**  
     
   어떤 기술이든 자유롭게 막 써볼 수 있는 환경이 중요하다. 이렇게도 써보고 저렇게도 써봐야 빨리 익숙해진다. 하지만 비용이 부담스러워서 EC2에서 자유롭게 쿠버네티스를 활용하기가 망설여질 것이다.

위 2가지 문제점을 해결할 수 있는 방법이 있다. 바로 **k3s**를 활용하는 것이다.

### **✅ k3s란?**

**k3s**란 **쿠버네티스(k8s)의 경량 버전**으로 설치가 간단하며, 컴퓨팅 리소스를 훨씬 적게 먹는다. 그리고 실제 프로덕션 환경에서 종종 사용하기도 한다. k3s를 활용하면 위에서 얘기한 2가지 단점을 해결할 수 있다.

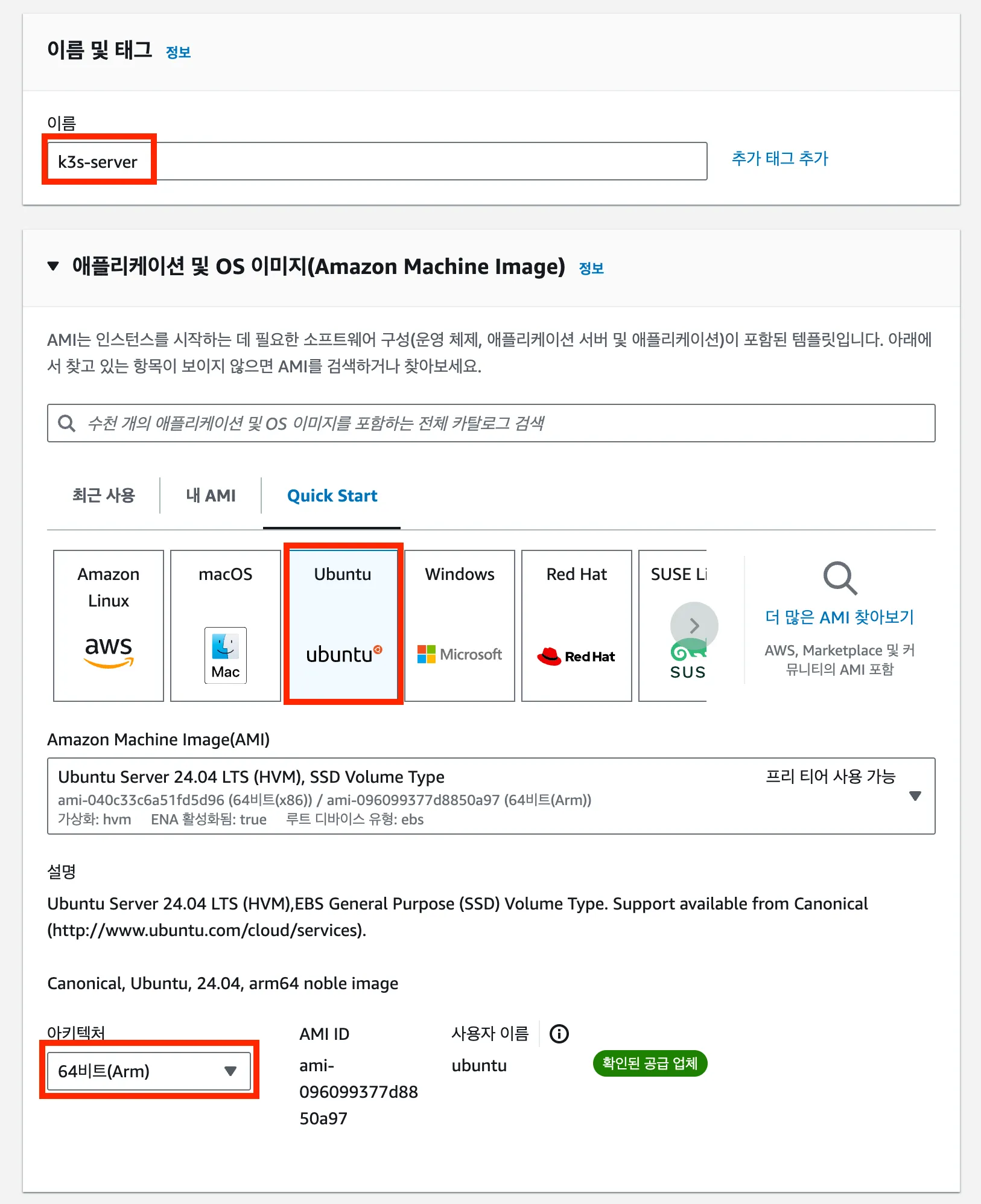
1. **EC2에서 도커/쿠버네티스 설치하기 (k3s)**

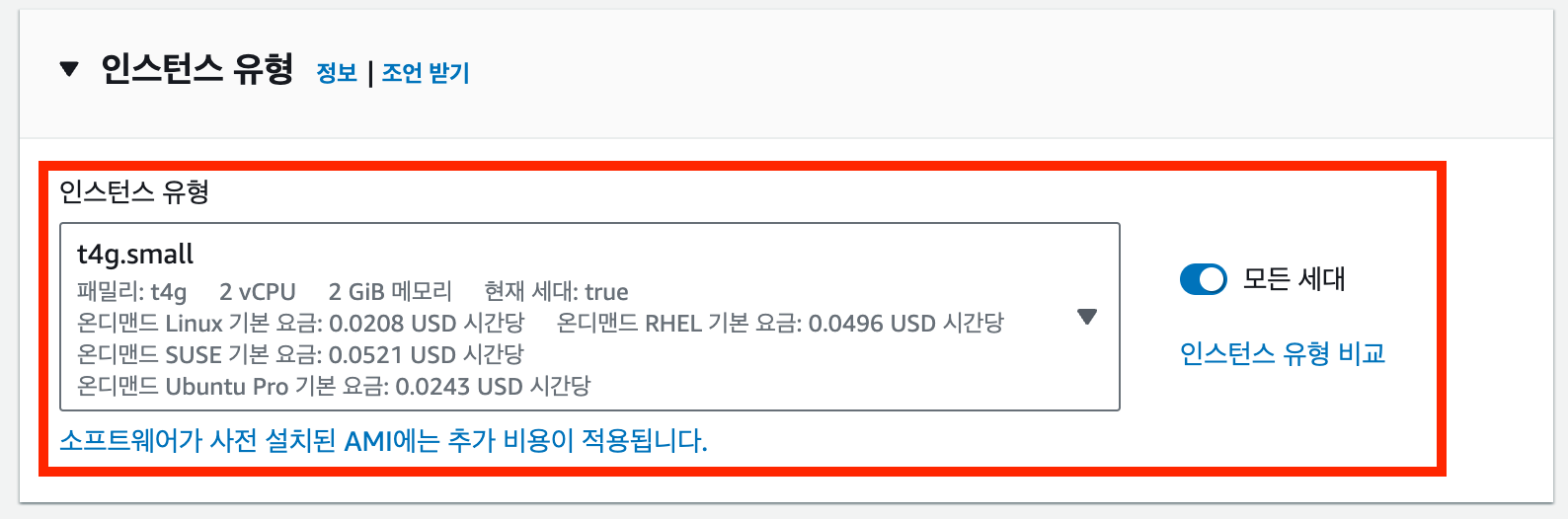
#### **✅ 설치 추천 사양**

**k3s의 권장 설치 사양이 CPU 2 core, RAM 1 GB이다. 백엔드 서버도 같이 실행시킬 것을 고려해 AWS EC2 인스턴스는 t4g.small로 생성하자.**

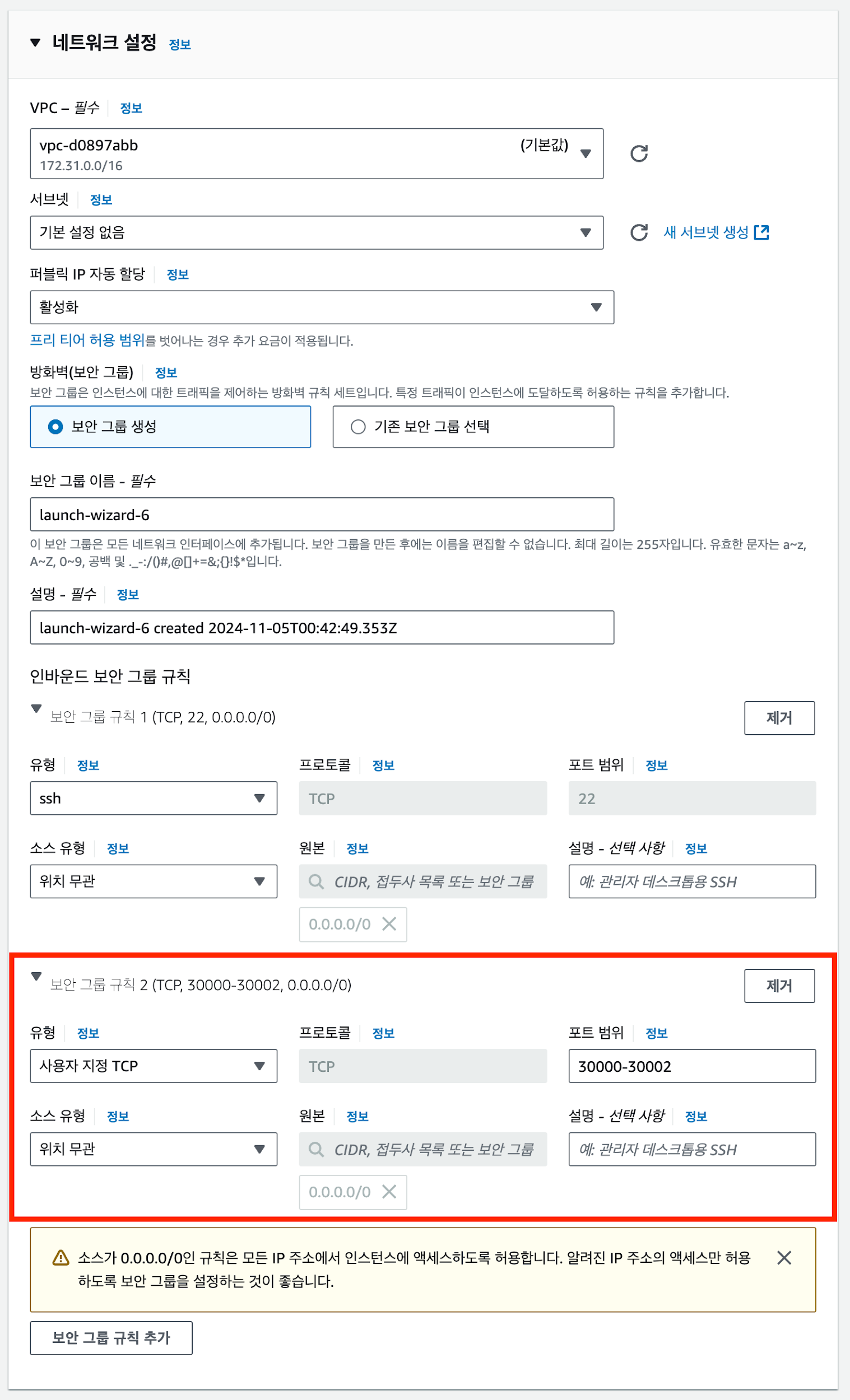
[**K3s is very lightweight, but has some minimum requirements as outlined below.**](https://docs.k3s.io/kr/installation/requirements)

#### **✅ EC2 인스턴스 생성하기**

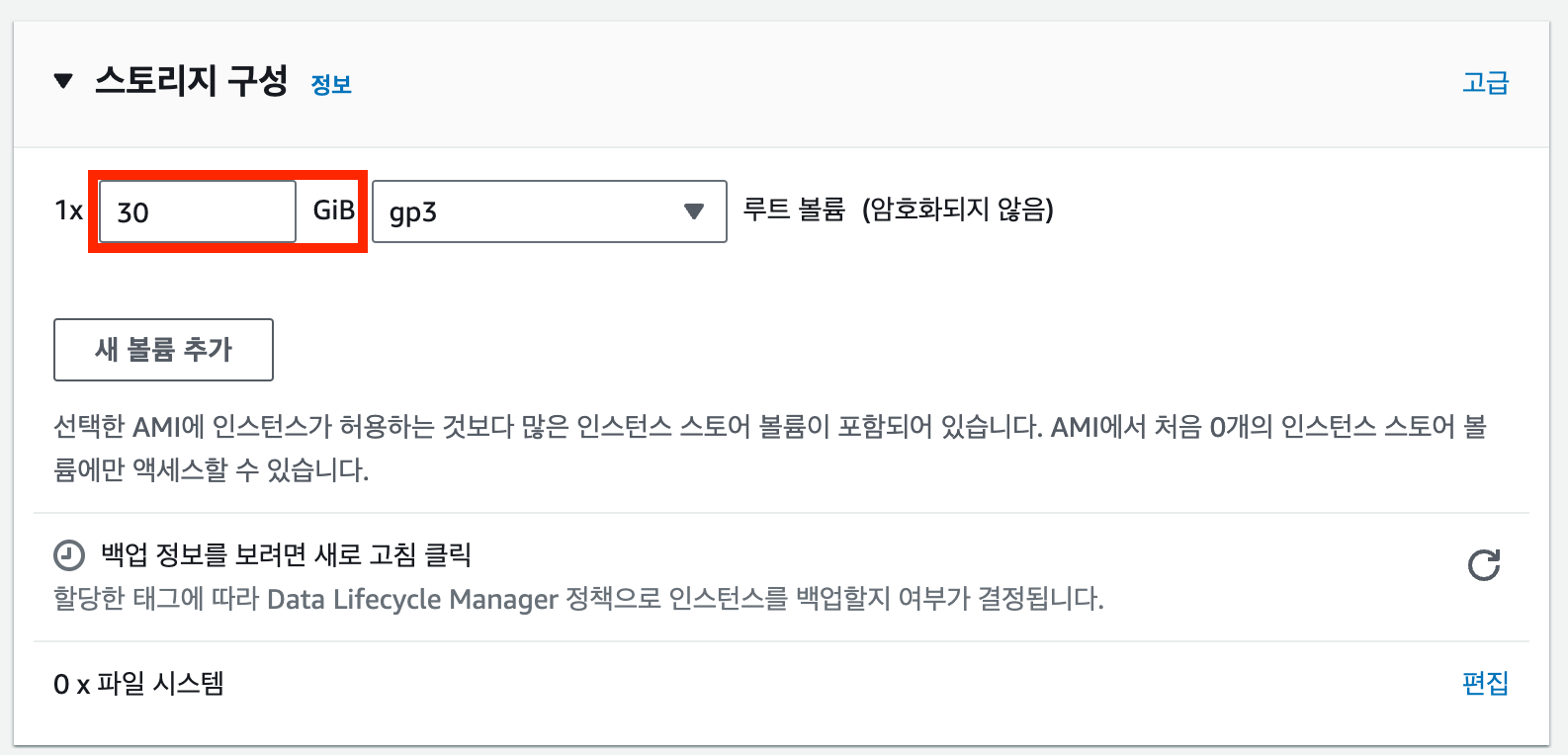
****

****

**ALT**

****

**ALT**

****

**ALT**

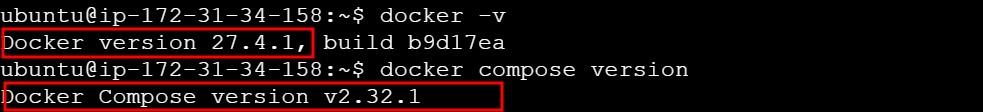
#### **✅ Docker 설치하기**

1. **설치하기**

| $ **sudo apt-get** update && \  **sudo apt-get install** -y apt-transport-https ca-certificates **curl** software-properties-common && \  **curl** -fsSL https://download.docker.com/linux/ubuntu/gpg | **sudo** apt-key **add** - && \  **sudo** apt-key fingerprint 0EBFCD88 && \  **sudo** add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" && \  **sudo apt-get** update && \  **sudo apt-get install** -y docker-ce && \  **sudo usermod** -aG **docker** ubuntu && \  newgrp **docker** && \  **sudo curl** -L "https://github.com/docker/compose/releases/download/2.27.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose && \  **sudo chmod** +x /usr/local/bin/docker-compose && \  **sudo ln** -s /usr/local/bin/docker-compose /usr/bin/docker-compose |
| --- |

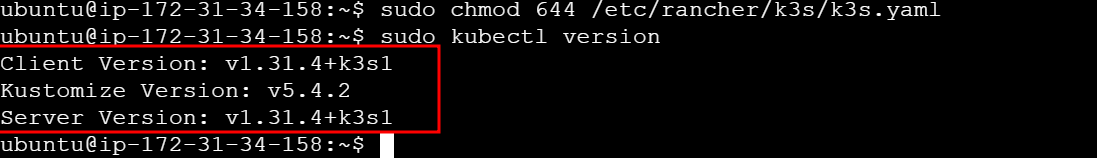
1. **잘 설치됐는 지 확인하기**

| $ docker -v # Docker 버전 확인  $ docker compose version # Docker Compose 버전 확인 |
| --- |

****

#### **✅ k3s 설치하기**

| $ **curl** -sfL https://get.k3s.io | **sh** - # k3s 설치  $ **sudo chmod** 644 /etc/rancher/k3s/k3s.yaml # 권한 부여  $ **sudo** kubectl version # k3s 잘 설치됐는 지 확인 |
| --- |

****

1. **[예제] 디플로이먼트, 서비스를 활용해 웹 서버(Nginx) 띄워보기**

#### **✅ 디플로이먼트, 서비스를 활용해 웹 서버(Nginx) 띄워보기**

1. **매니페스트 파일 작성하기**

$ vi deployment.yaml

| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: nginx-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3  selector:  matchLabels:  app: nginx # 아래에서 정의한 Pod 중 'app: nginx'이라는 값을 가진 파드를 선택    # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: nginx  spec:  containers:  - name: nginx-container # 컨테이너 이름  image: nginx # 컨테이너를 생성할 때 사용할 이미지  ports:  - containerPort: **80** # 컨테이너에서 사용하는 포트를 명시적으로 표현 |
| --- |

****

service.yaml

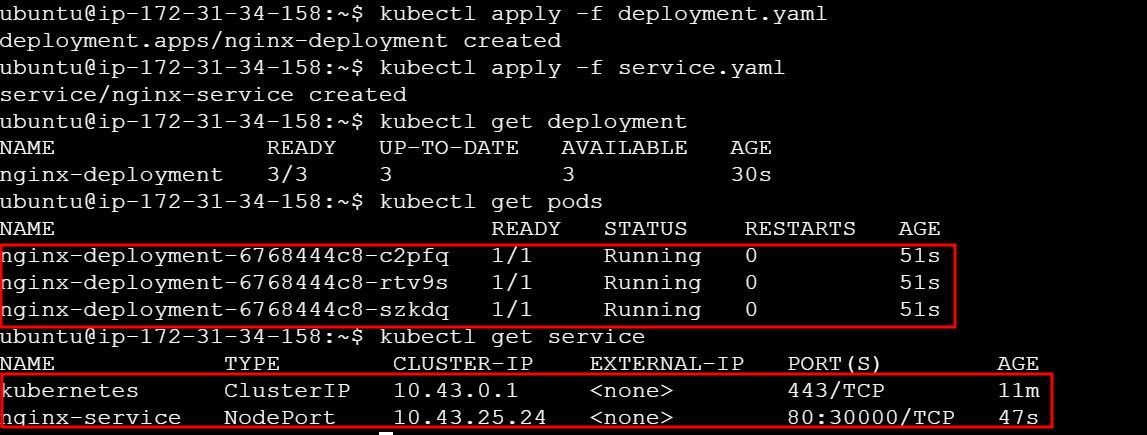
| apiVersion: v1  kind: Service  # Service 기본 정보  metadata:  name: nginx-service # Service 이름    # Service 세부 정보  spec:  type: NodePort # Service의 종류  selector:  app: nginx # 실행되고 있는 파드 중 'app: nginx'이라는 값을 가진 파드와 서비스를 연결  ports:  - protocol: TCP # 서비스에 접속하기 위한 프로토콜  port: **80** # 쿠버네티스 내부에서 Service에 접속하기 위한 포트 번호  targetPort: **80** # 매핑하기 위한 파드의 포트 번호  nodePort: **30000** # 외부에서 사용자들이 접근하게 될 포트 번호 |
| --- |

1. **매니페스트 파일 실행시키기**

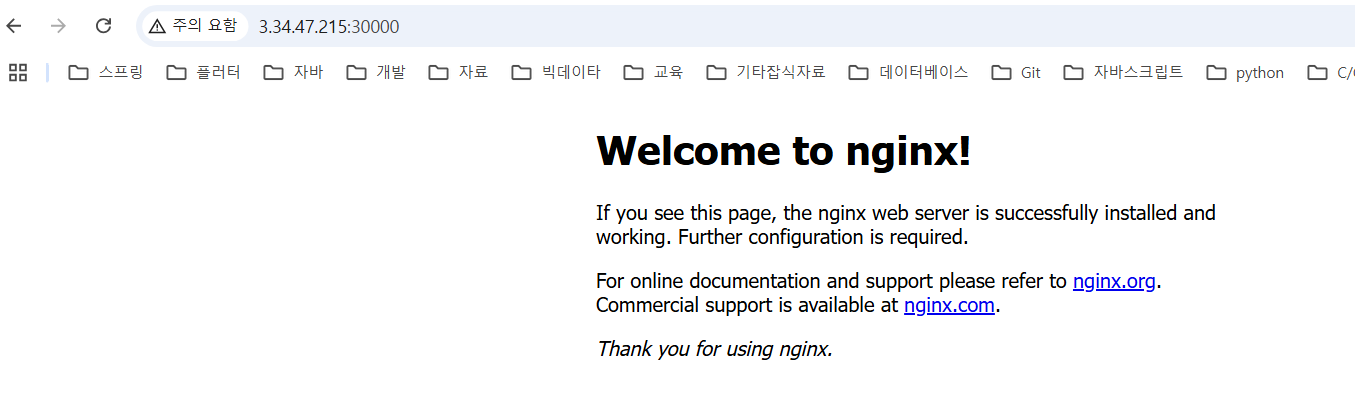
| $ kubectl apply -f deployment.yaml  $ kubectl apply -f service.yaml |
| --- |

1. **오브젝트가 작 생성 됐는지 확인**

| $ kubectl get pods  $ kubectl get deployment  $ kubectl get **service** |
| --- |

****

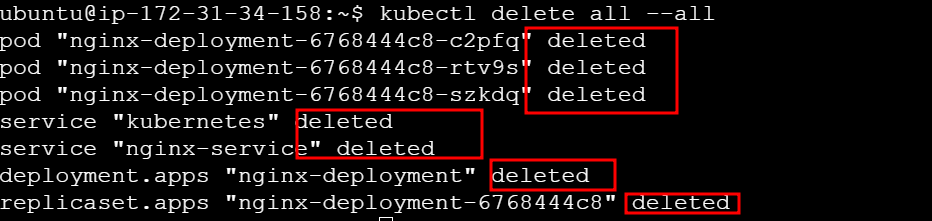
1. **정상적으로 잘 접속되는 지 확인하기**

****

**ALT**

#### **✅ 생성한 오브젝트 정리하기**

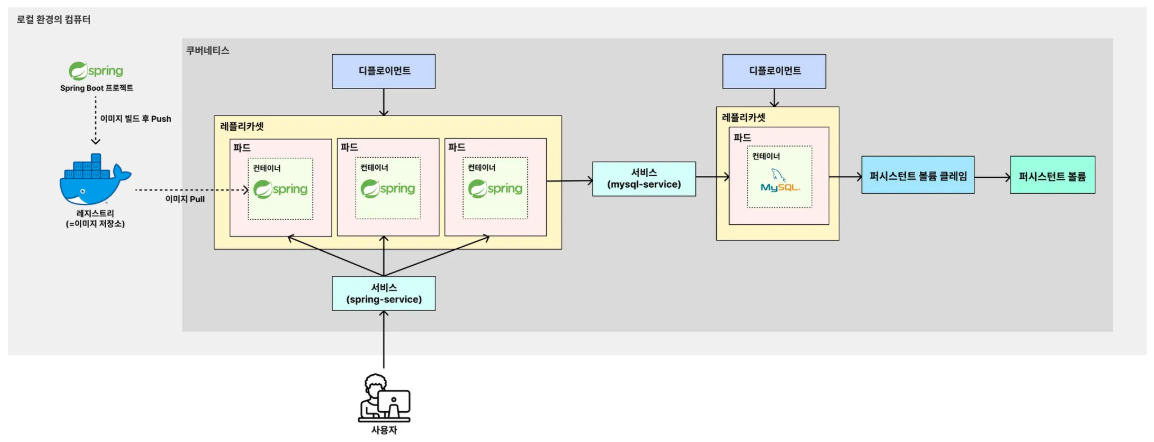
| $ kubectl delete all --all |
| --- |

****

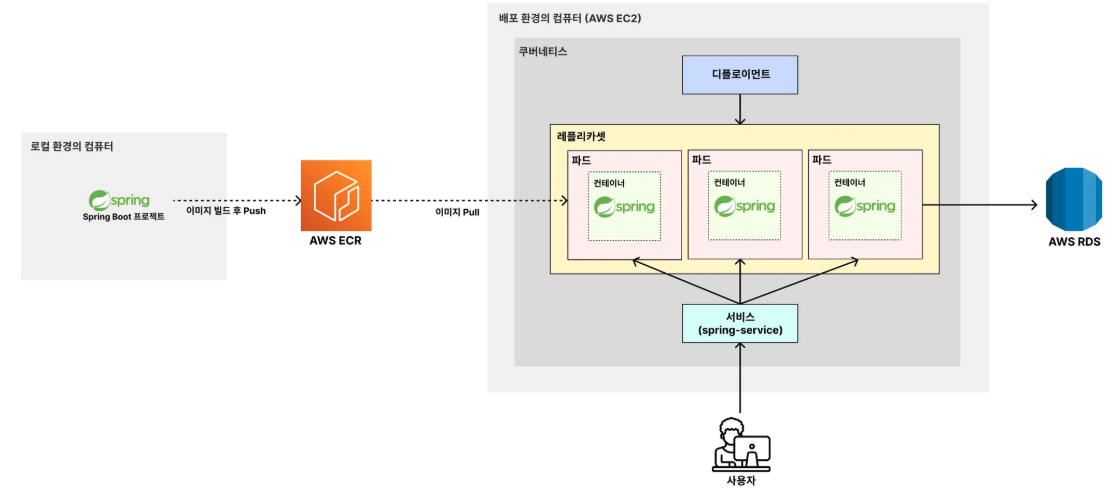
1. **[예제] 백엔드(Spring Boot) 서버 배포하기, DB 연동하기 (+ RDS, ECR)**

🧑EC2에 k3s를 활용해 백엔드(Spring Boot) 서버를 배포하고, 백엔드 서버에 DB(MySQL)도 연동시켜보자.

### **✅ 로컬 환경에서의 아키텍처**



### **✅ 배포 환경에서의 아키텍처**



로컬 환경에서의 아키텍처와의 차이점은 크게 2가지이다.

1. **로컬에 도커 이미지를 저장하지 않고, 외부 저장소인 AWS ECR에 도커 이미지를 저장한다.**
2. **로컬의 데이터베이스를 사용하지 않고, 외부 데이터베이스인 AWS RDS를 활용한다.**

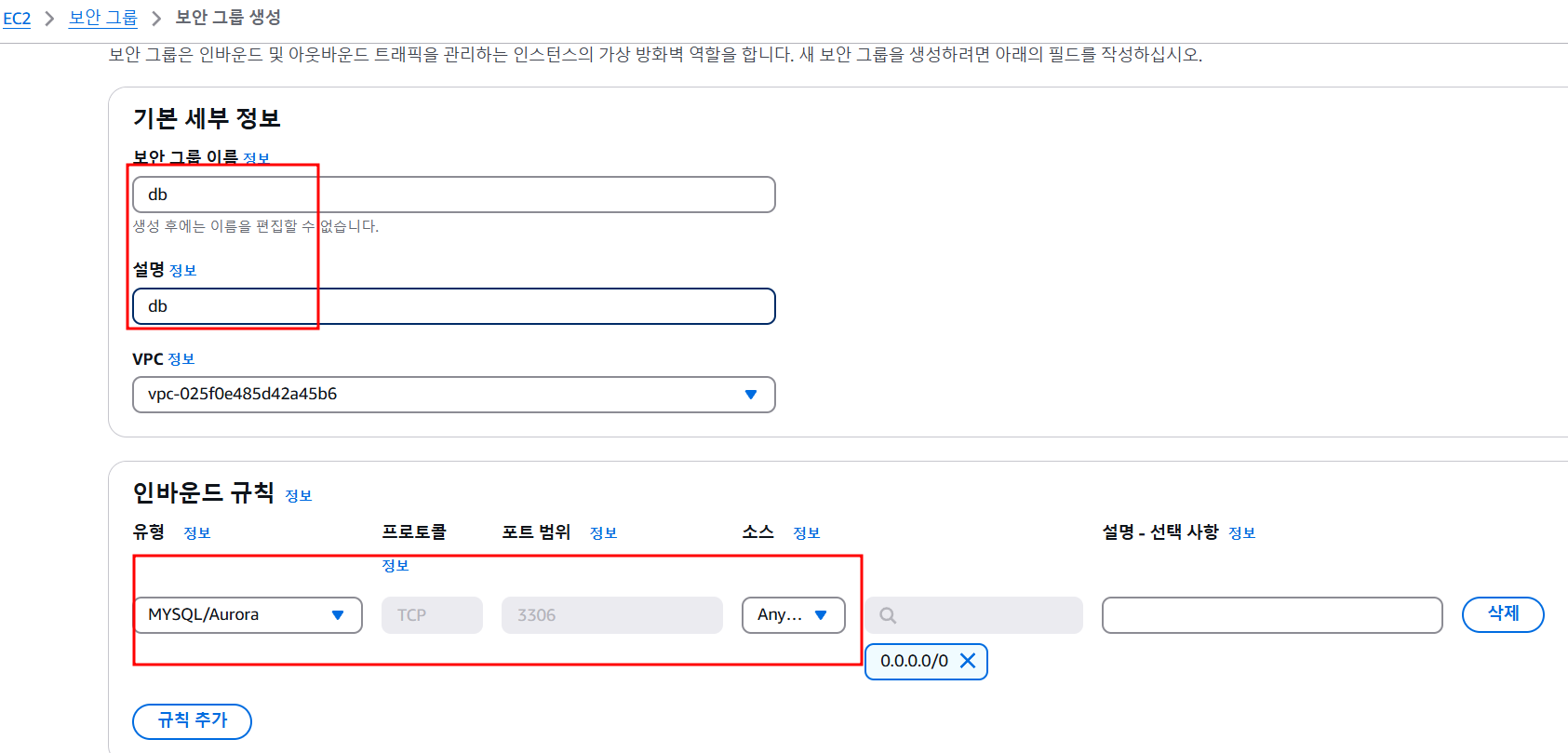
위의 아키텍처로 환경을 구성해보자.

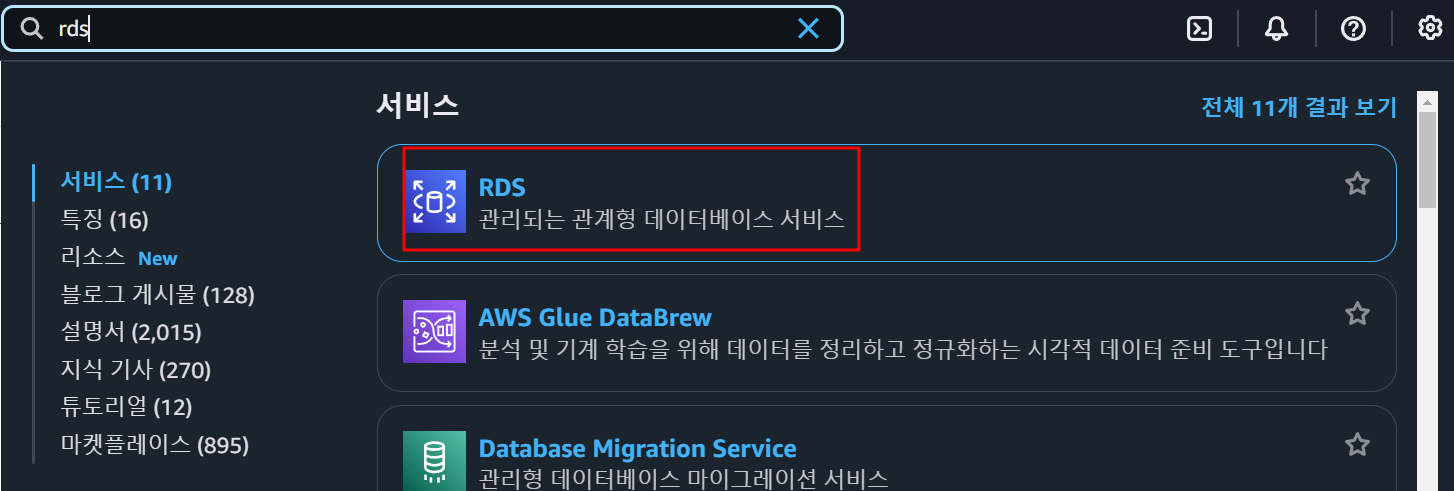
1. **RDS, ECR 생성하기**

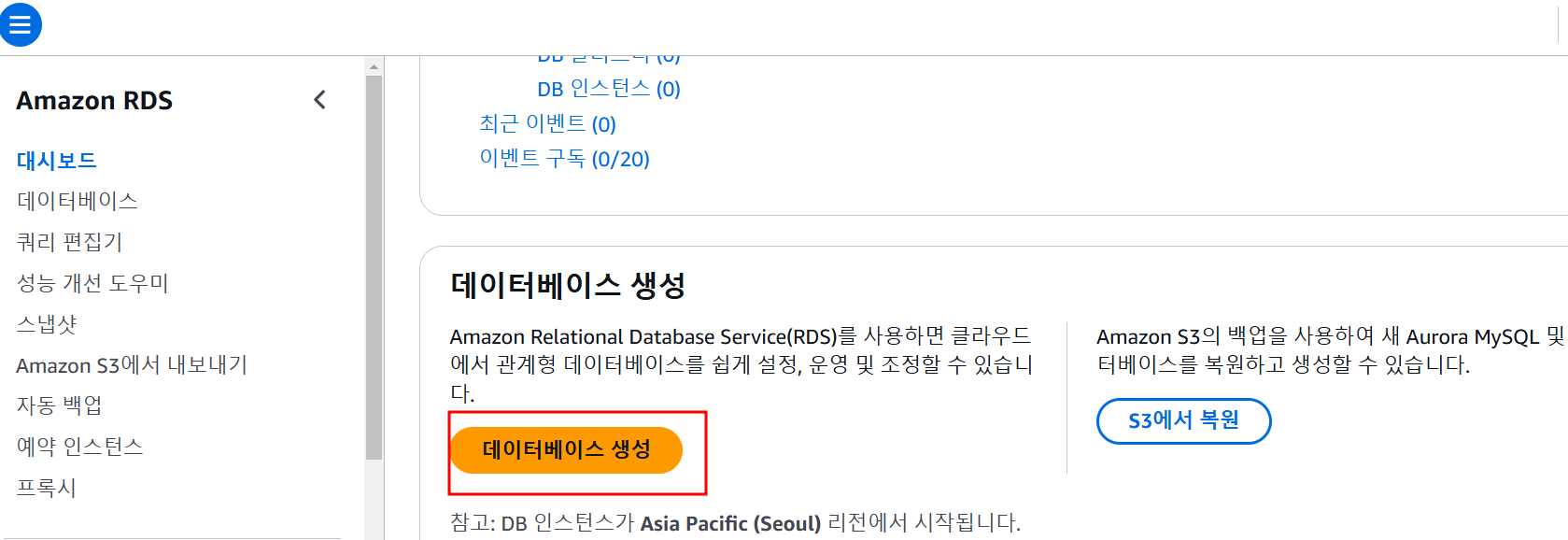
### **✅ RDS 생성하기**

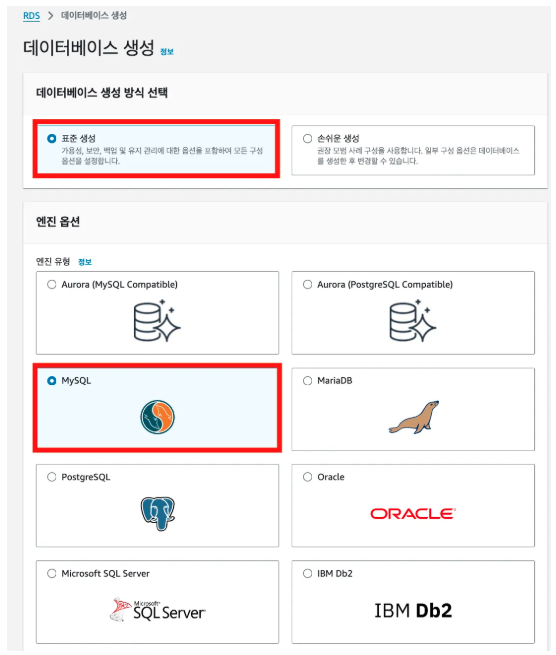
1. **RDS 생성하기**

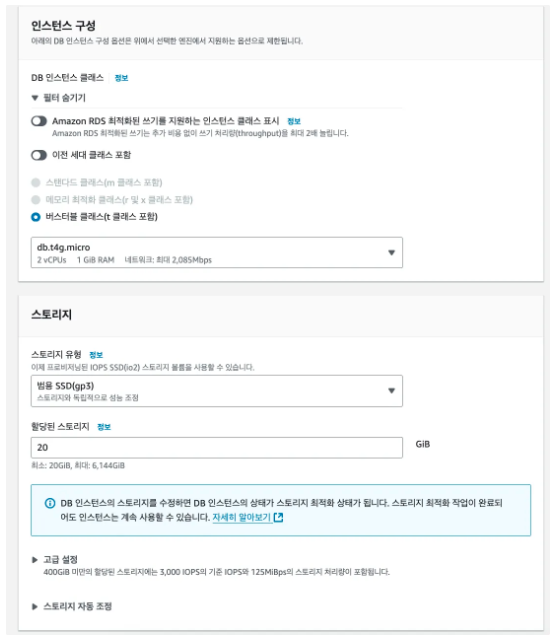
**새로운 창에서 새로운 보안그룹 생성 한 후 다음 진행**

****

****

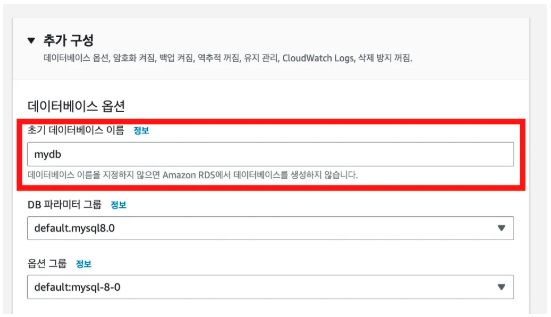
****

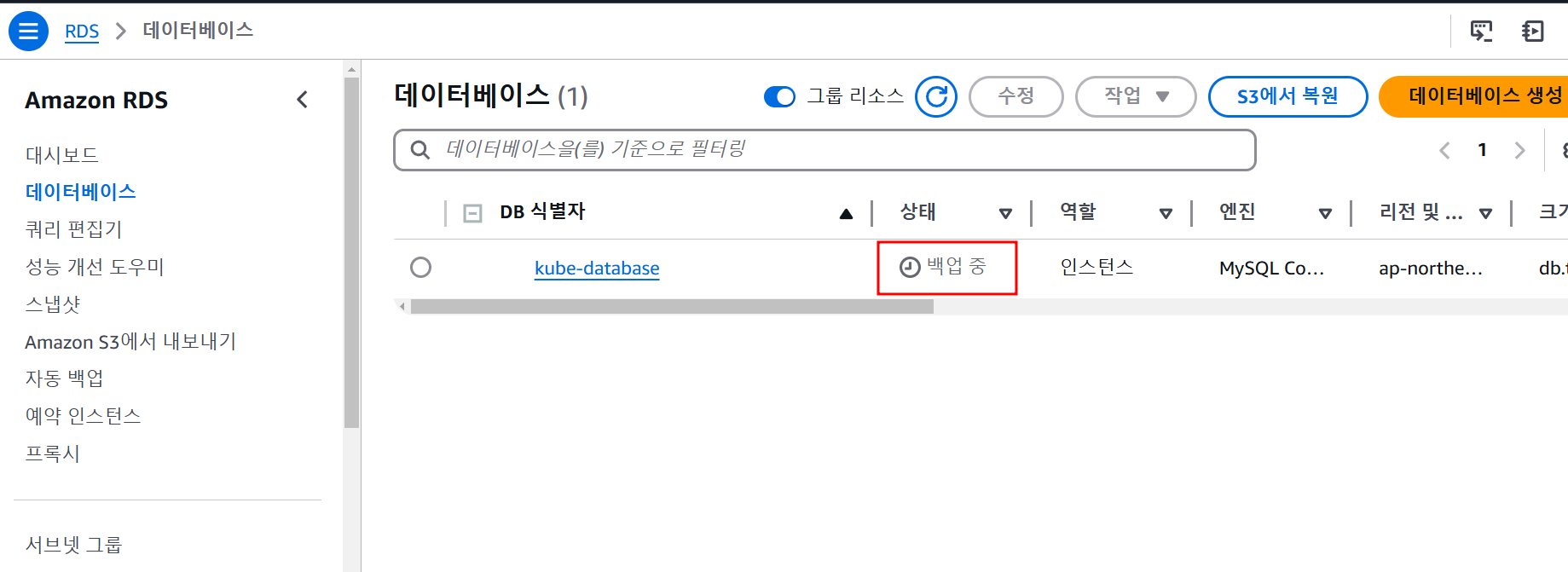
****

**  
  
  
**

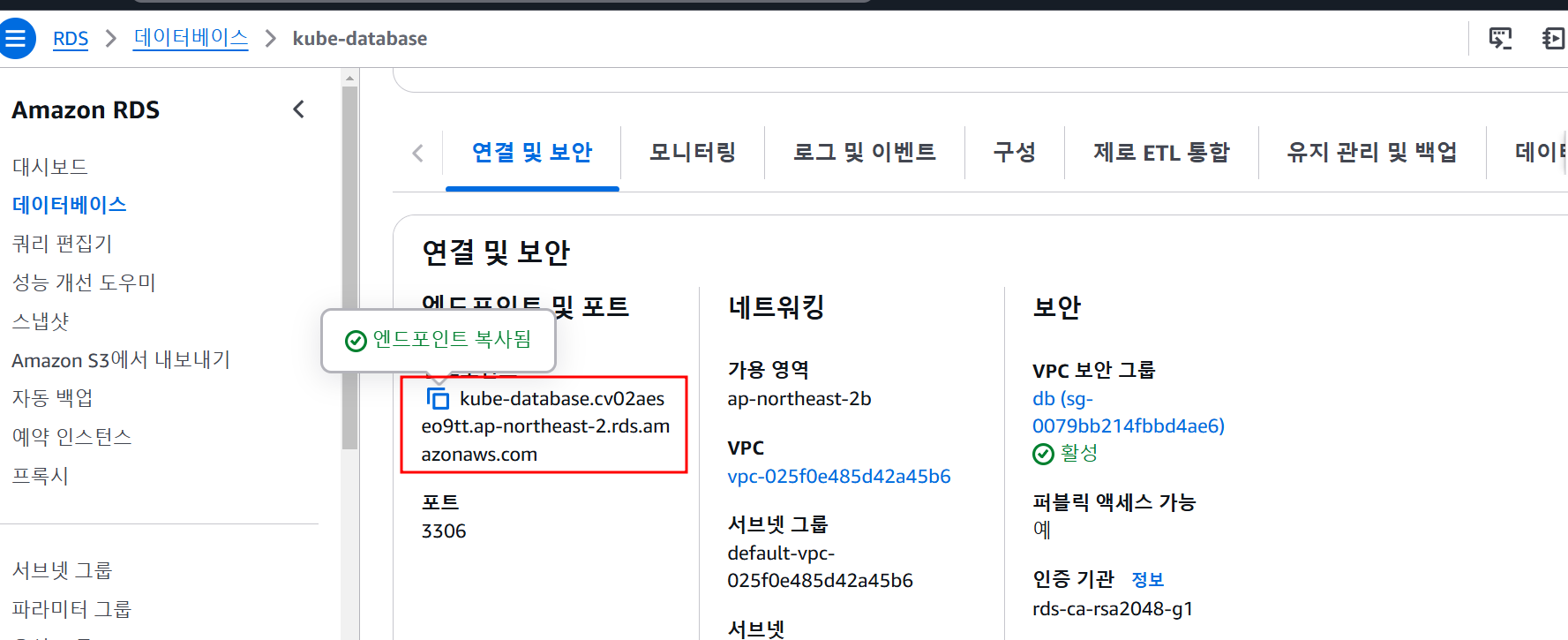
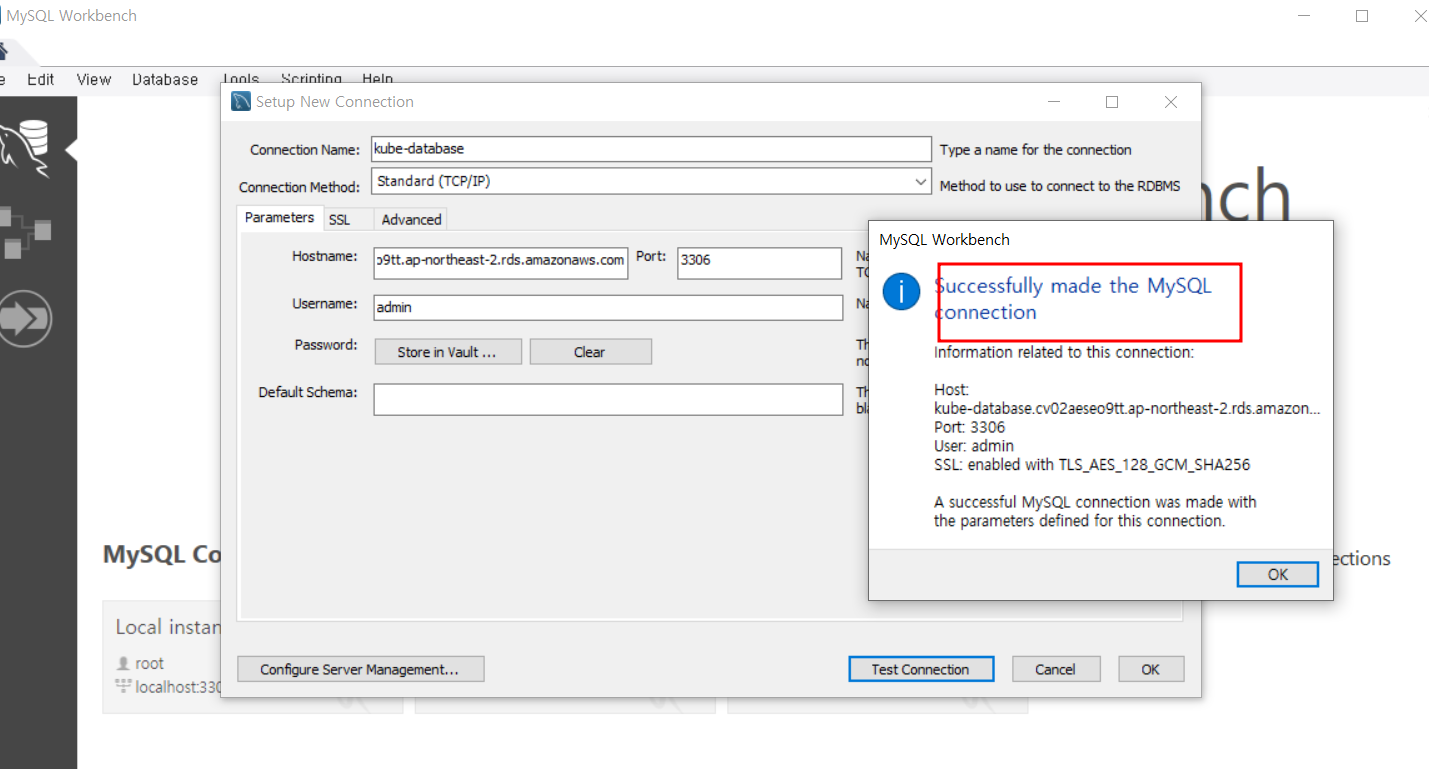
****

* + EC2 페이지에서 보안그룹 생성 한 뒤에 **기존 VPC 보안 그룹** 선택하기

****  (나머지 옵션은 디폴트 그대로)



생성됨.

1. **잘 접속되는 지 확인하기  
     
   **

### 

### **✅ ECR 생성하기**

****

1. **백엔드(Spring Boot) 서버 프로젝트, 매니페스트 파일 구성하기**

### **✅ 새로운 아키텍처에 맞게 코드 수정하기**

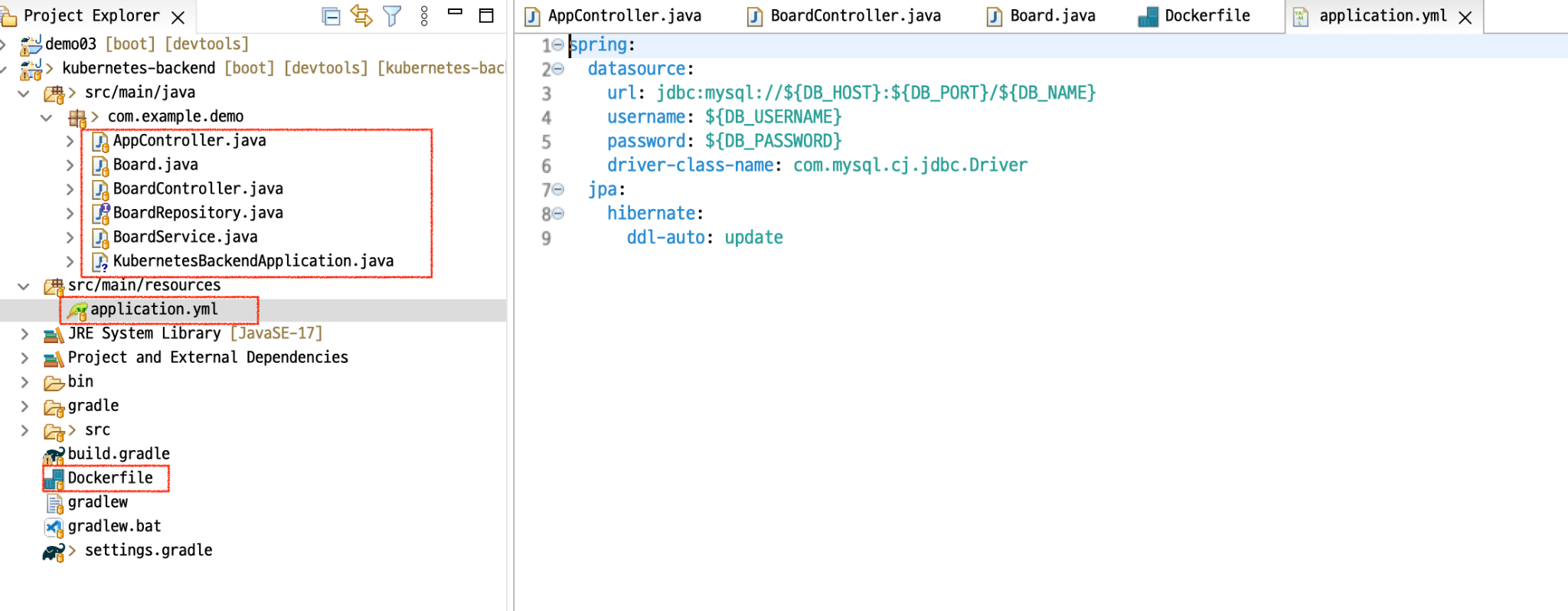
1. **Spring Boot 프로젝트 클론 받기**  
   로컬 환경에서 Spring Boot 프로젝트를 구현했다고 가정하고, 편의를 위해 완성된 코드를 클론 받자.

| $ git clone https://github.com/JSCODE-COURSE/kubernetes-backend.git |
| --- |

1. **쿠버네티스 매니페스트 파일 클론 받기**  
   로컬 환경에서 매니페스트 파일을 직접 작성했다고 가정하고, 편의를 위해 완성된 코드를 클론 받자.

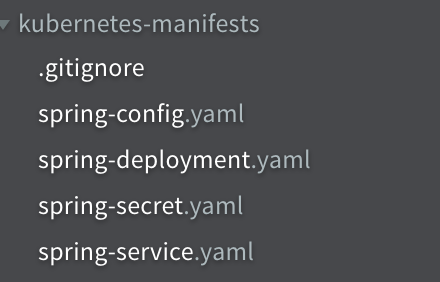
| $ **git** clone https://github.com/JSCODE-COURSE/kubernetes-manifests.git |
| --- |

1. **코드 살펴보기**  
    **[Spring Boot 프로젝트]**
   * **/boards GET** : 게시글 조회 API
   * **/boards POST** : 게시글 작성 API
   * JPA를 활용해 DB와 연동하고 있는 상태
   * Docker 이미지로 빌드할 수 있게 **Dockerfile**을 작성해놓은 상태



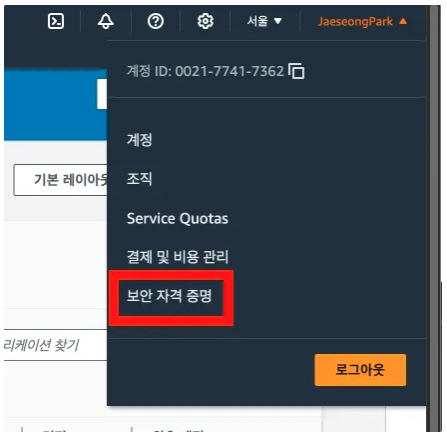
**[매니페스트 파일]**

* + Spring Boot 서버를 띄우는 데 필요한 **Deployment, Service, ConfigMap, Secret** 파일들로 구성되어 있음
  + **spring-secret.yaml**은 민감한 값이 포함된 파일이므로, 원래라면 **.gitignore**에 추가해서 버전 관리가 되지 않게 만들어야 한다. 하지만 편의상 Github Repository에 같이 올려두었다.
    - **spring-secret.yaml**은 서버에 따로 전달해서 활용하거나, [SealedSecret](https://coffeewhale.com/sealedsecret)을 활용해서 관리하는 편이다. (이 외에도 AWS Secrets Manager, Vault 등 시크릿 값을 관리하는 다양한 서비스가 있다.)

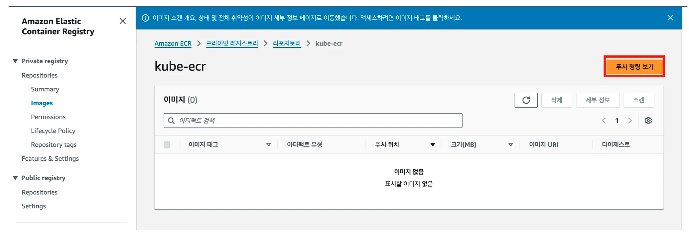


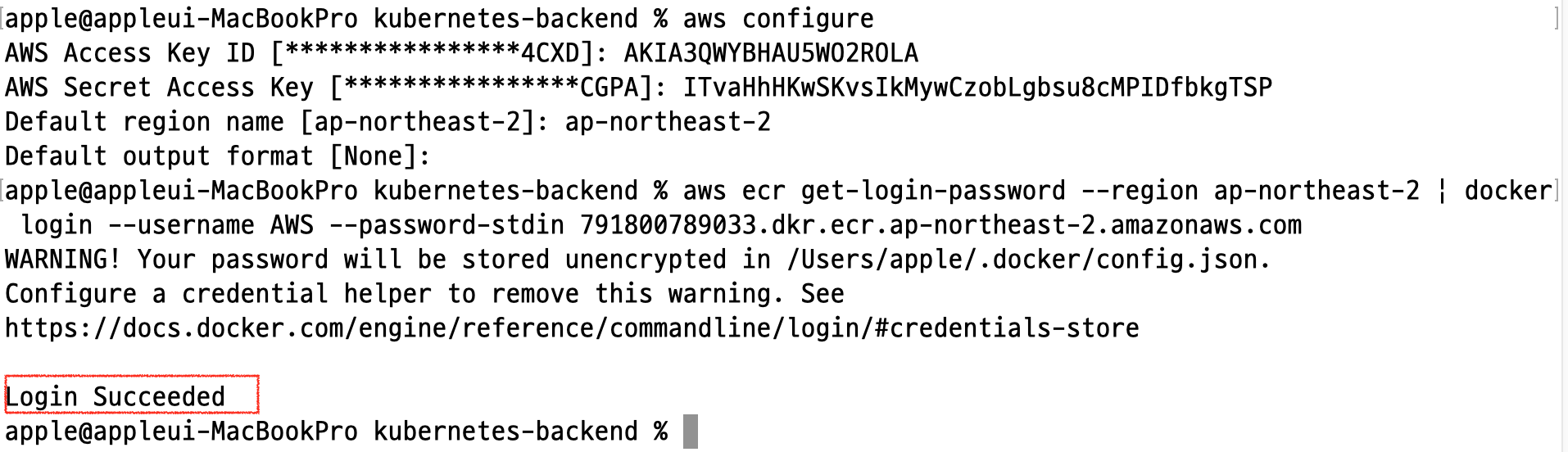
1. **백엔드(Spring Boot) 서버 빌드한 후 ECR로 Push하기**

### **✅ 백엔드(Spring Boot) 서버 빌드한 후 ECR로 Push하기**

1. **로컬 환경에 AWS CLI 설치하기**  
     
    [[AWS] 📚 AWS CLI 설치 & 등록 방법 - 쉽고 빠르게 설명](https://inpa.tistory.com/entry/AWS-%F0%9F%93%9A-AWS-CLI-%EC%84%A4%EC%B9%98-%EC%82%AC%EC%9A%A9%EB%B2%95-%EC%89%BD%EA%B3%A0-%EB%B9%A0%EB%A5%B4%EA%B2%8C)
2. **Access Key 발급받기**  
     
     
    발급받은 Access Key와 Secret Access Key를 잘 보관해두기 바란다. (유출되면 안 되니 조심해서 관리하자)
3. **AWS CLI로 액세스 키 등록하기**

| $ aws configure  AWS Access Key ID [None]: **<위에서 발급한 Key id>**  AWS Secret Access Key [None]: **<위에서 발급한 Secret Access Key>**  Default region name [None]: **ap-northeast-2**  Default output **format** [None]: |
| --- |

1. **ECR로 들어가서 생성한 레포지토리로 들어가기**  
   
2. **푸시 명령 확인하기**  
   

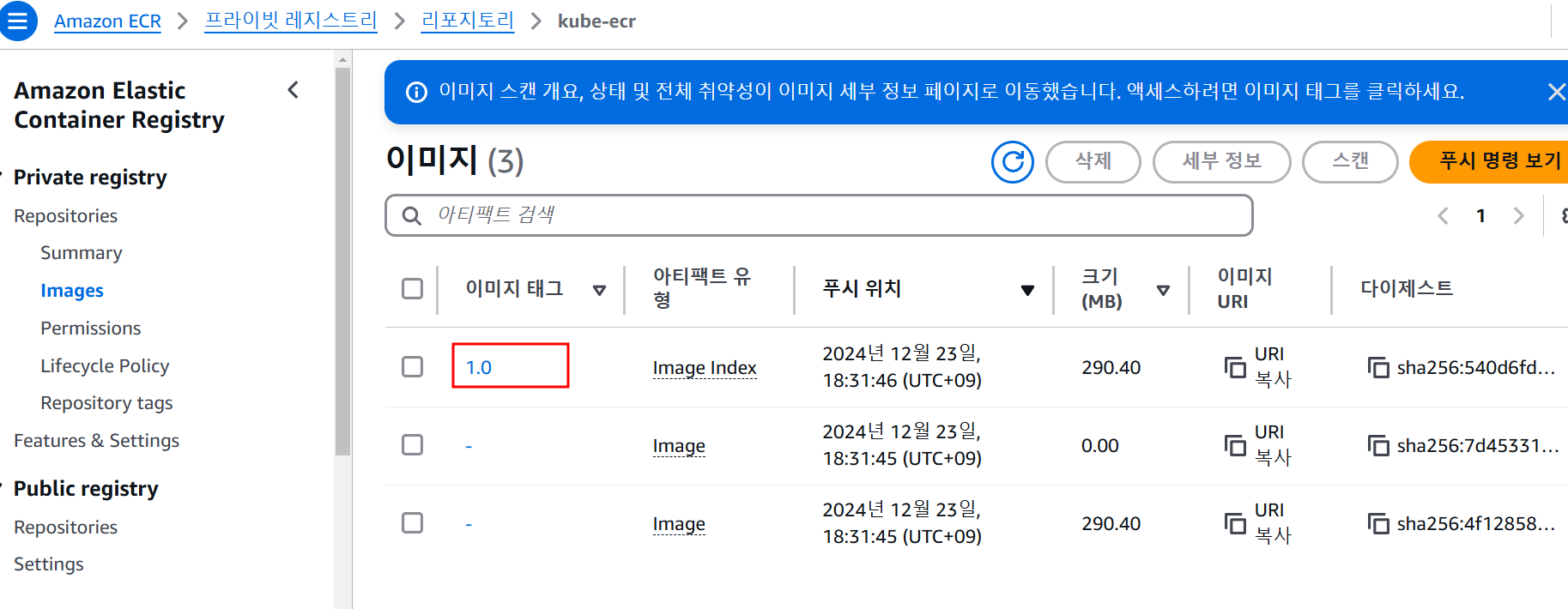


1. **로컬 환경에서 Spring Boot 프로젝트를 Docker 이미지로 빌드한 후에 ECR로 Push하기**

| $ ./gradlew clean build  $ docker build -t kube-ecr .  $ docker tag kube-ecr:latest 002177417362.dkr.ecr.ap-northeast-2.amazonaws.com/kube-ecr:**1.0**  $ docker push 002177417362.dkr.ecr.ap-northeast-2.amazonaws.com/kube-ecr:**1.0** |
| --- |



* + 주의) 태그명에 **latest**를 쓰지 말고 구체적인 버전명을 작성하기

1. **정상적으로 이미지가 Push 됐는 지 확인하기**  
   
2. **EC2가 ECR로부터 이미지를 Pull 받아올 수 있게 권한 부여하기**

#### **✅ EC2가 ECR로부터 이미지를 Pull 받아올 수 있게 권한 부여하기**

1. **AWS CLI 설치하기**

| $ cd ~  $ sudo apt install unzip  $ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" -o "awscliv2.zip"  $ unzip awscliv2.zip  $ sudo ./aws/install  $ aws --version # 잘 출력된다면 정상 설치된 상태 |
| --- |

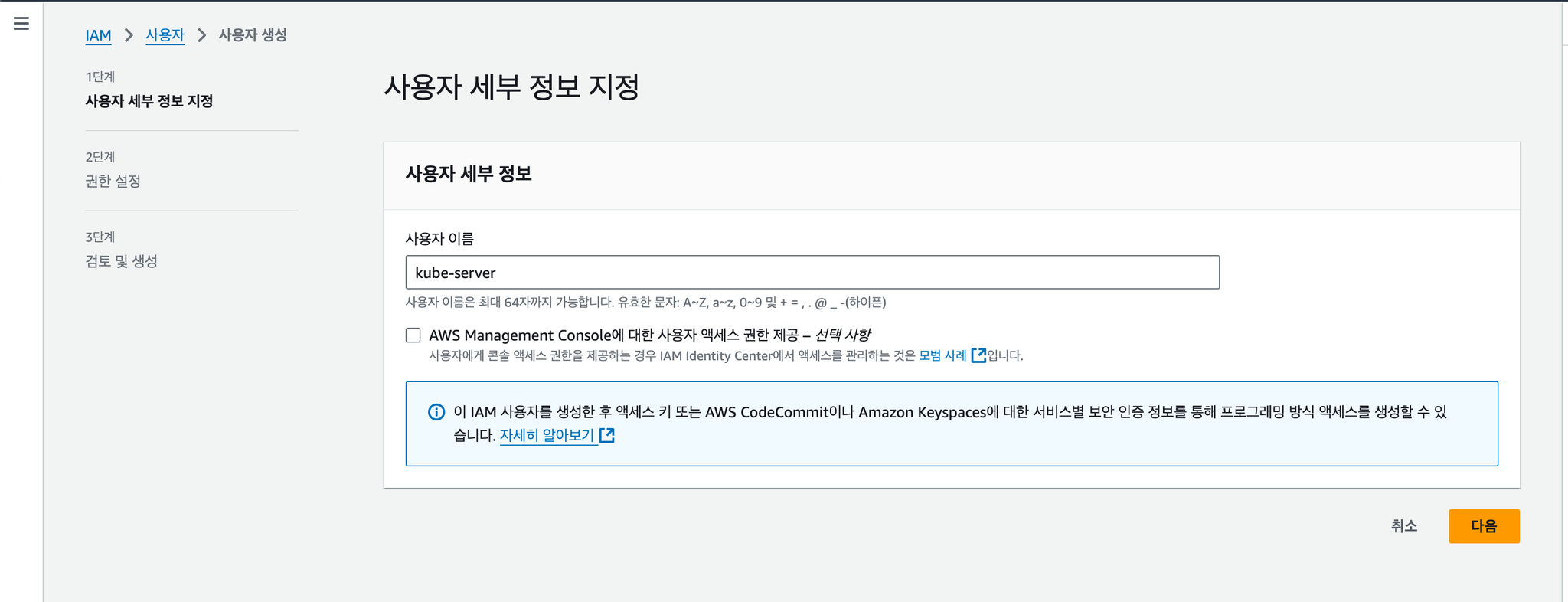
****

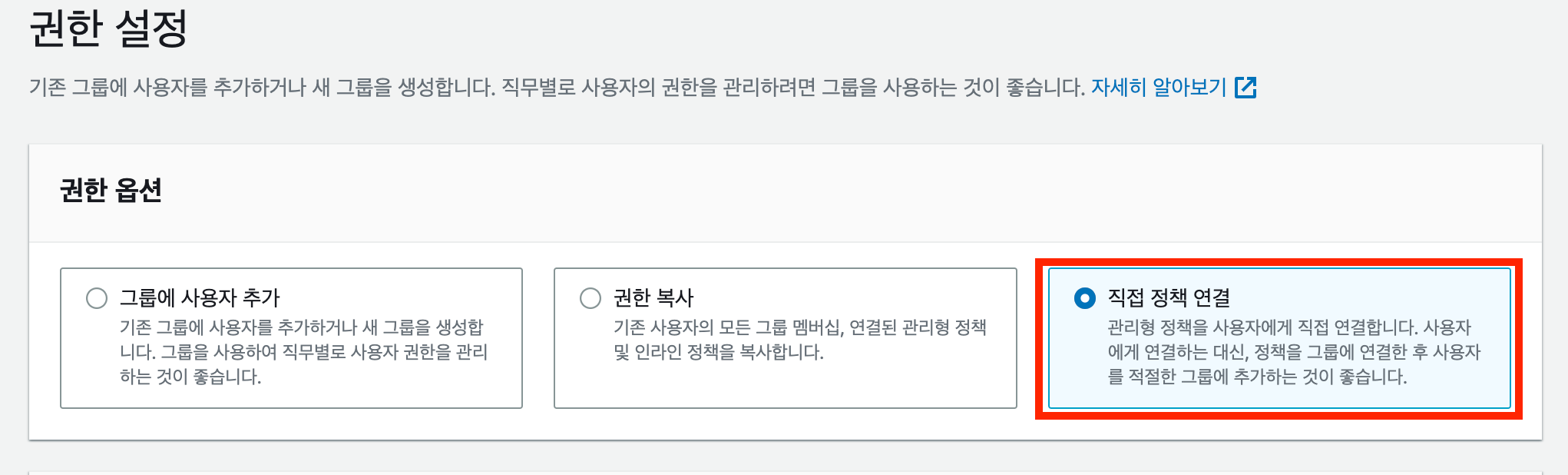
**다른버전 curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"**

[Installing or updating to the latest version of the AWS CLI -(참고문서)](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html)

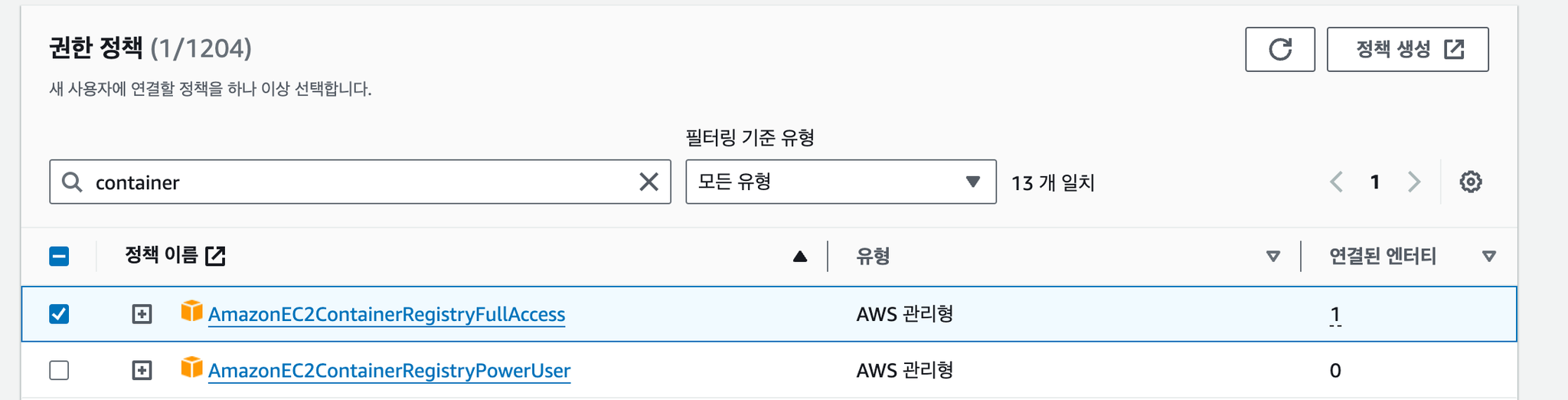
1. **IAM 사용자 만들기**



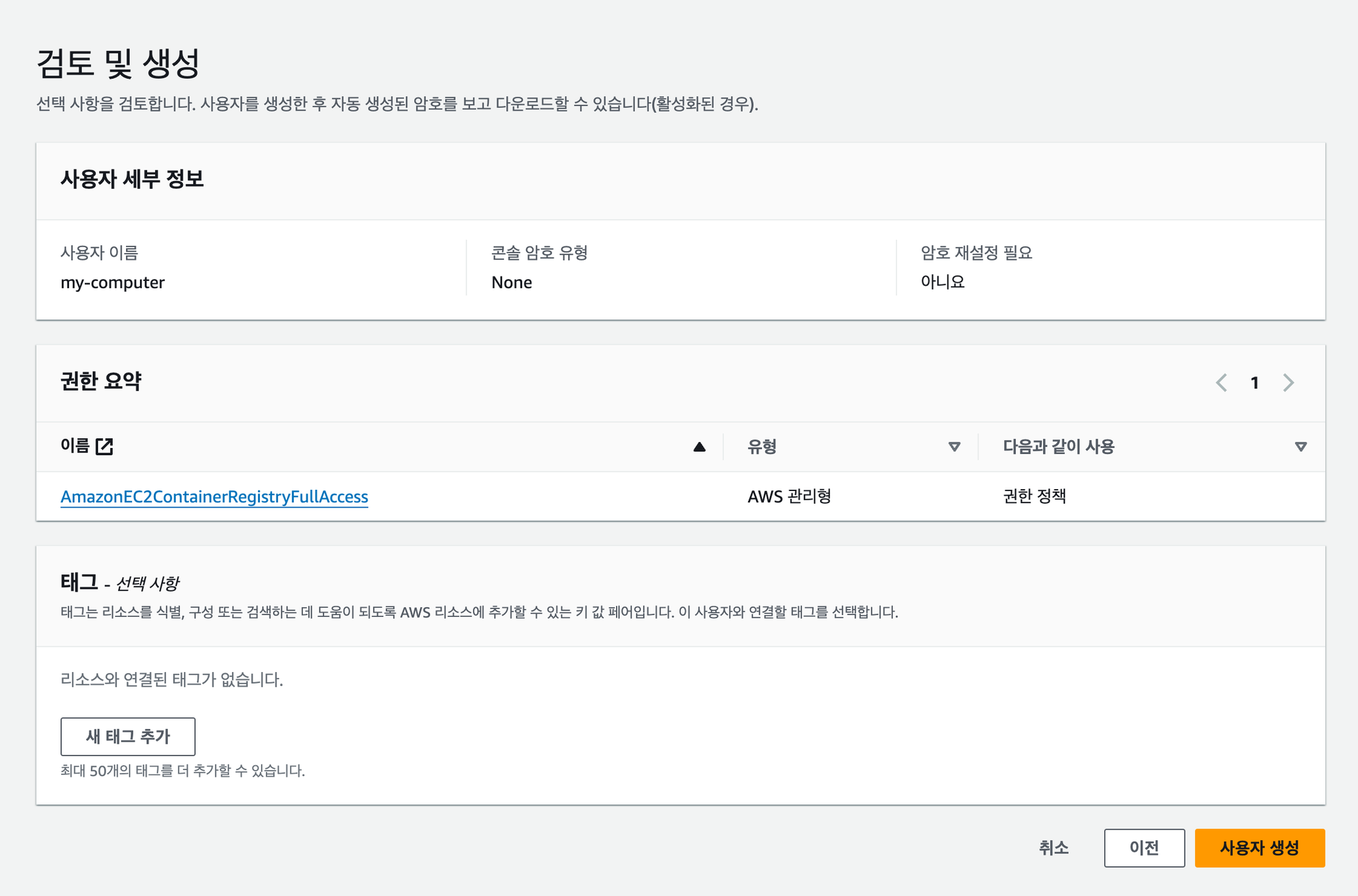
**A **

**AL **

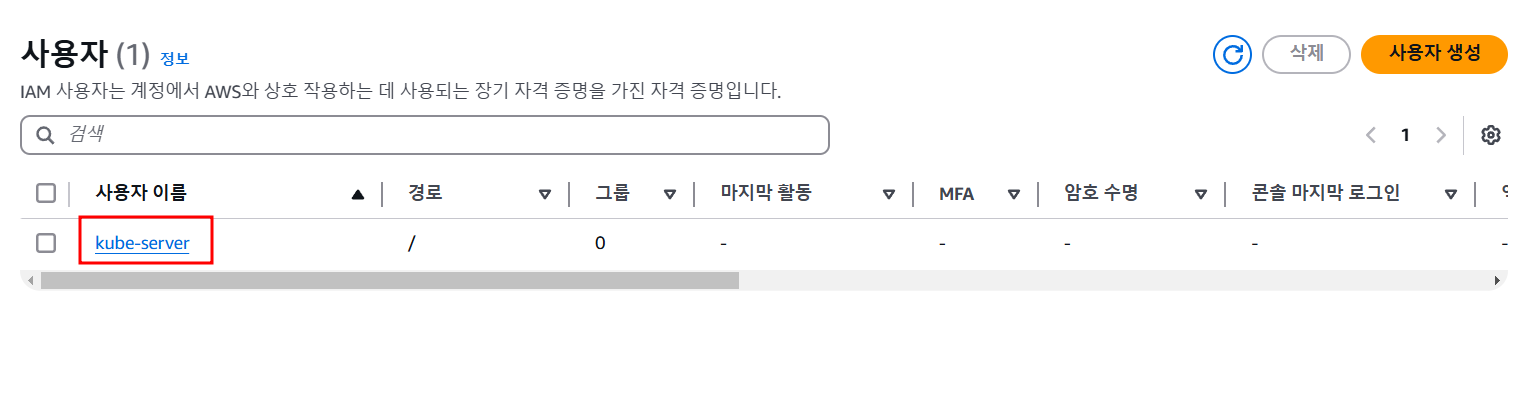
**ALT**

****

**ALT**

****

1. **Access Key 발급받기**

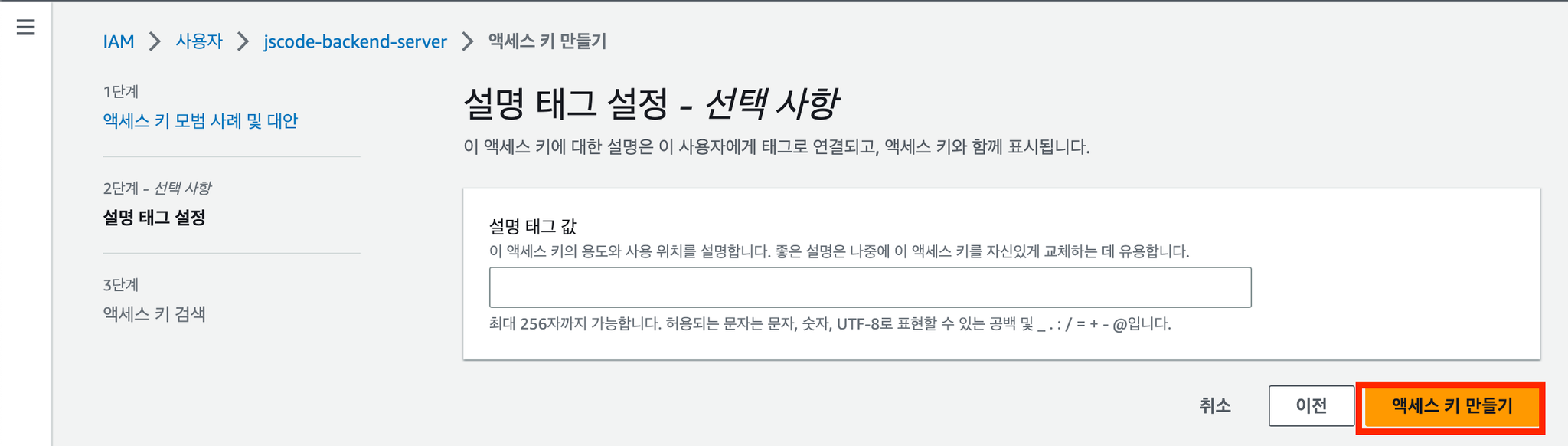
****

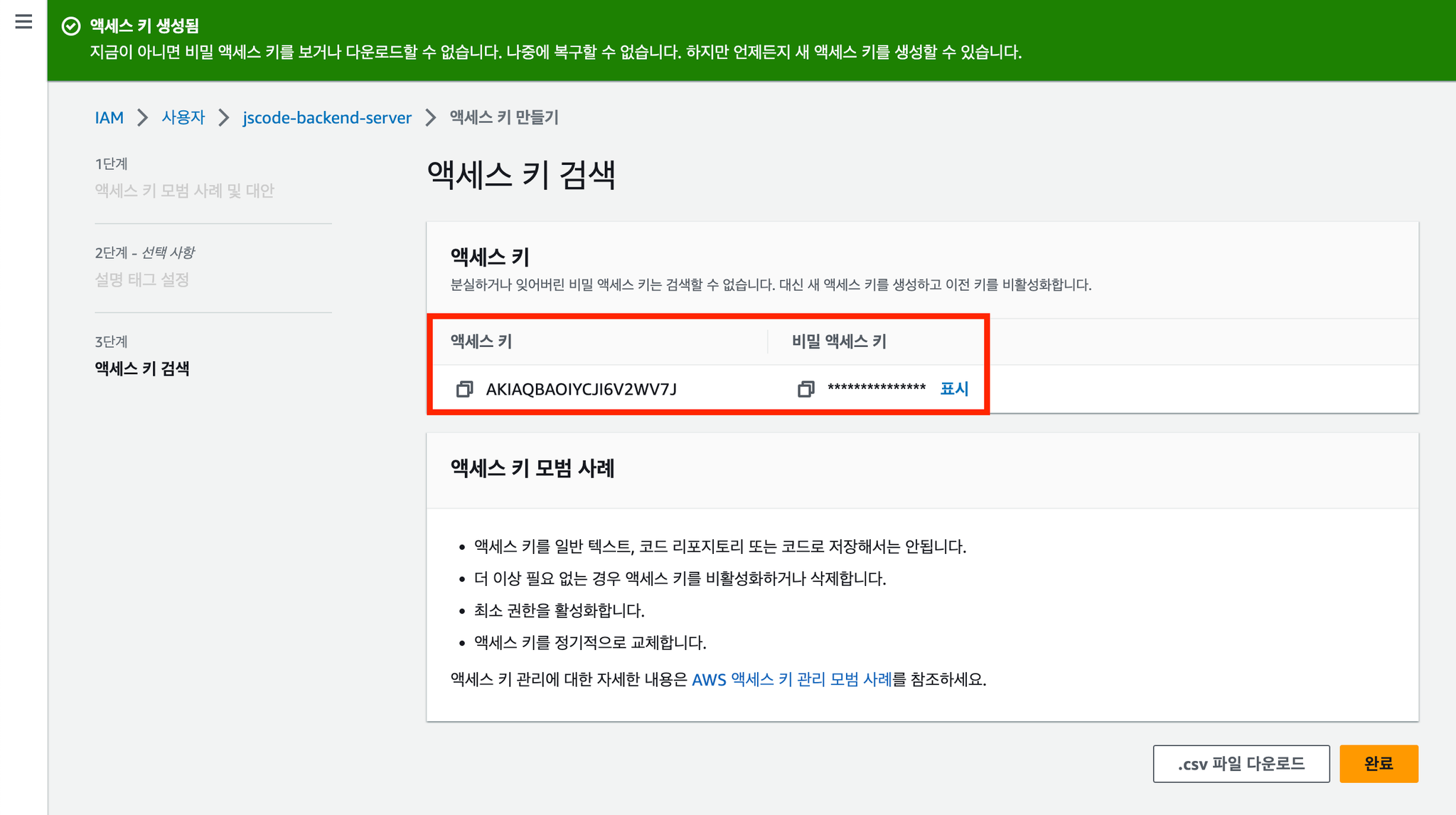
****



**ALT**

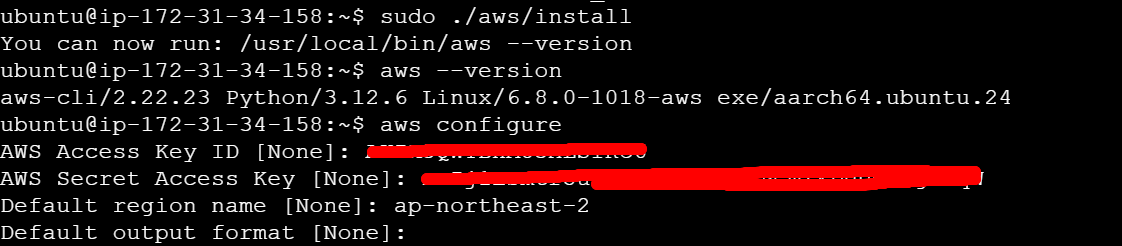
****

**A**

**ALT**

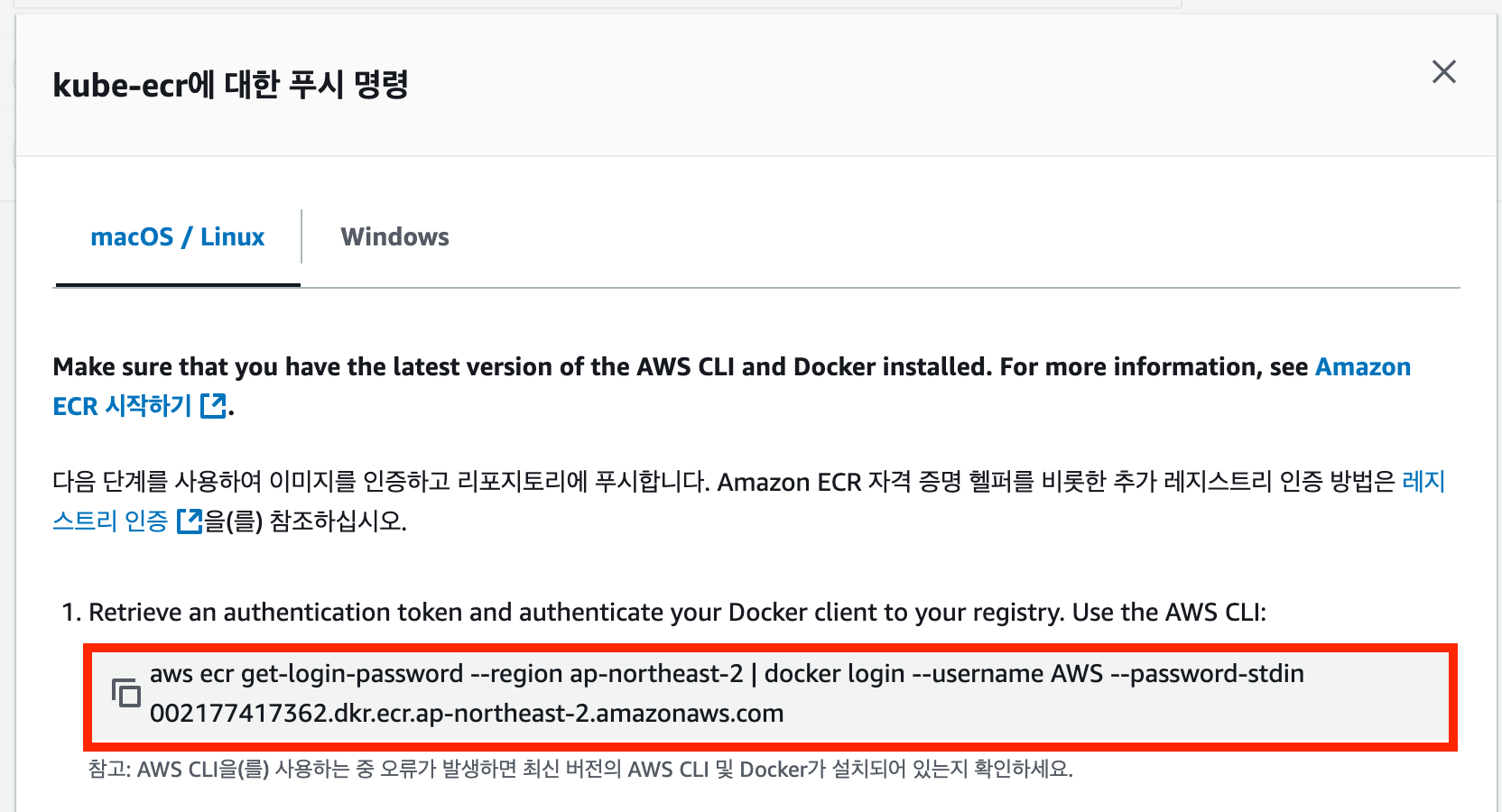
1. **AWS CLI로 액세스 키 등록하기**

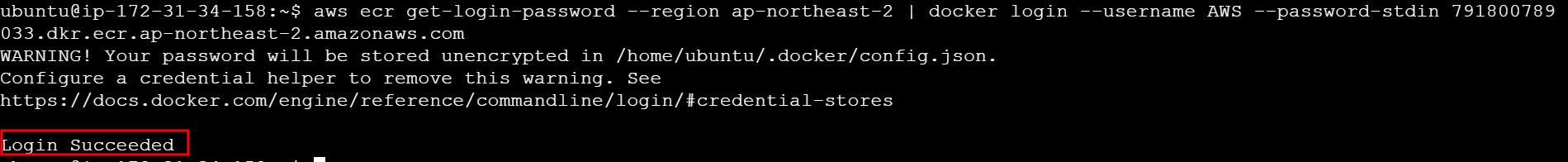
| **$ aws configure**  **AWS Access Key ID [None]:** <위에서 발급한 Key id>  **AWS Secret Access Key [None]:** <위에서 발급한 Secret Access Key>  **Default region name [None]:** ap-northeast-2  **Default output format [None]:** |
| --- |

**​ **

1. **ECR에 접근할 수 있는 권한 인증하기**

ECR의 레포지토리 페이지로 들어가서 푸시 명령에서 첫 번째 명령어를 복사한 뒤에 EC2에서 실행시키자.

****

****

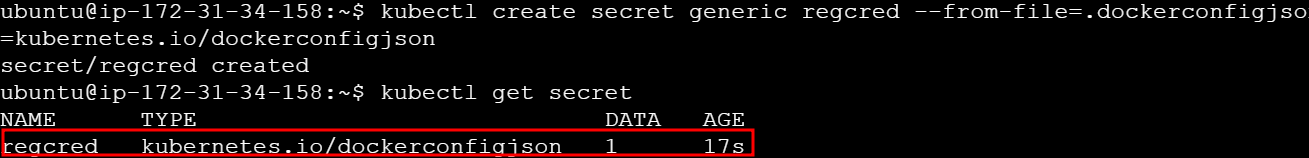
1. **쿠버네티스가 ECR로부터 이미지를 받아올 때 인증을 할 수 있도록 Secret 추가하기**

| $ kubectl create secret generic regcred --from-file=.dockerconfigjson=/home/ubuntu/.docker/config.json --type=kubernetes.io/dockerconfigjson |
| --- |

* **kubectl create secret** : 시크릿(Secret) 만드는 명령어
* **generic** : 시크릿(Secret)의 타입 중 하나 (= **Opaque**)
* **regcred** : 시크릿(Secret) 이름

1. **잘 생성 됐는지 확인**

| $ kubectl get secret |
| --- |

****

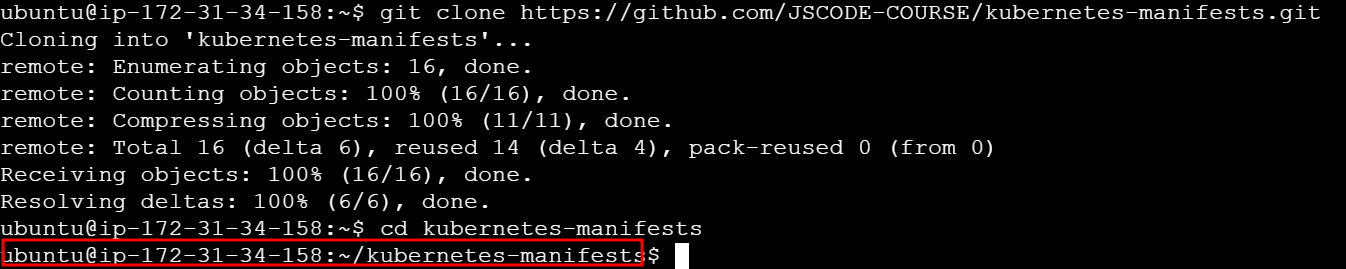
1. **EC2에서 쿠버네티스로 백엔드(Spring Boot) 서버 띄우기**

### **✅ EC2에서 쿠버네티스로 백엔드(Spring Boot) 서버 띄우기**

1. **매니페스트 파일 클론 받기**

AWS EC2에서 백엔드(Spring Boot) 서버를 띄울 때 오로지 매니페스트 파일만 있으면 된다. Spring Boot에 관련된 프로젝트 파일은 필요 없다. 왜냐하면 매니페스트 파일을 통해 백엔드(Spring Boot)서버를 실행시킬 이미지를 다운받아와서 실행시길 것이기 때문이다.

| $ git clone https://github.com/JSCODE-COURSE/kubernetes-manifests.git  $ cd kubernetes-manifests |
| --- |



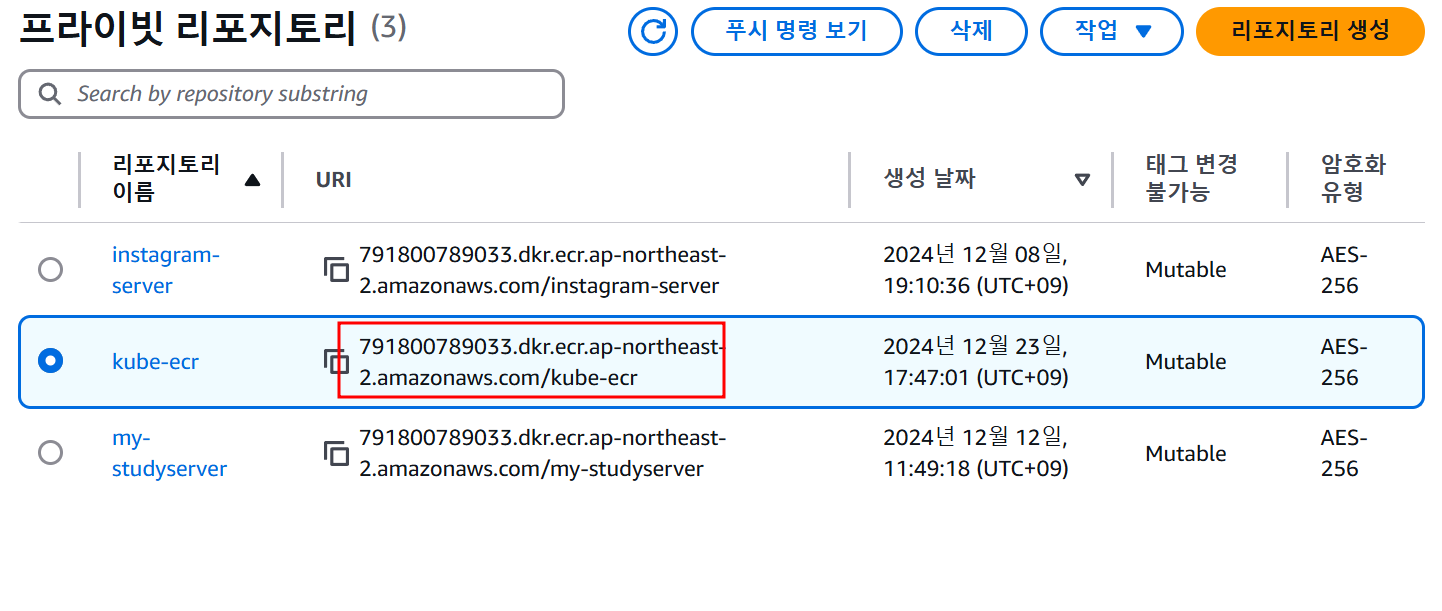
1. Deployment, ConfigMap, Secret 파일 수정하기

이전에 생성한 RDS, ECR 정보에 맞게 Deployment, ConfigMap, Secret 정보를 수정해주어야 한다.

spring-deployment.yaml

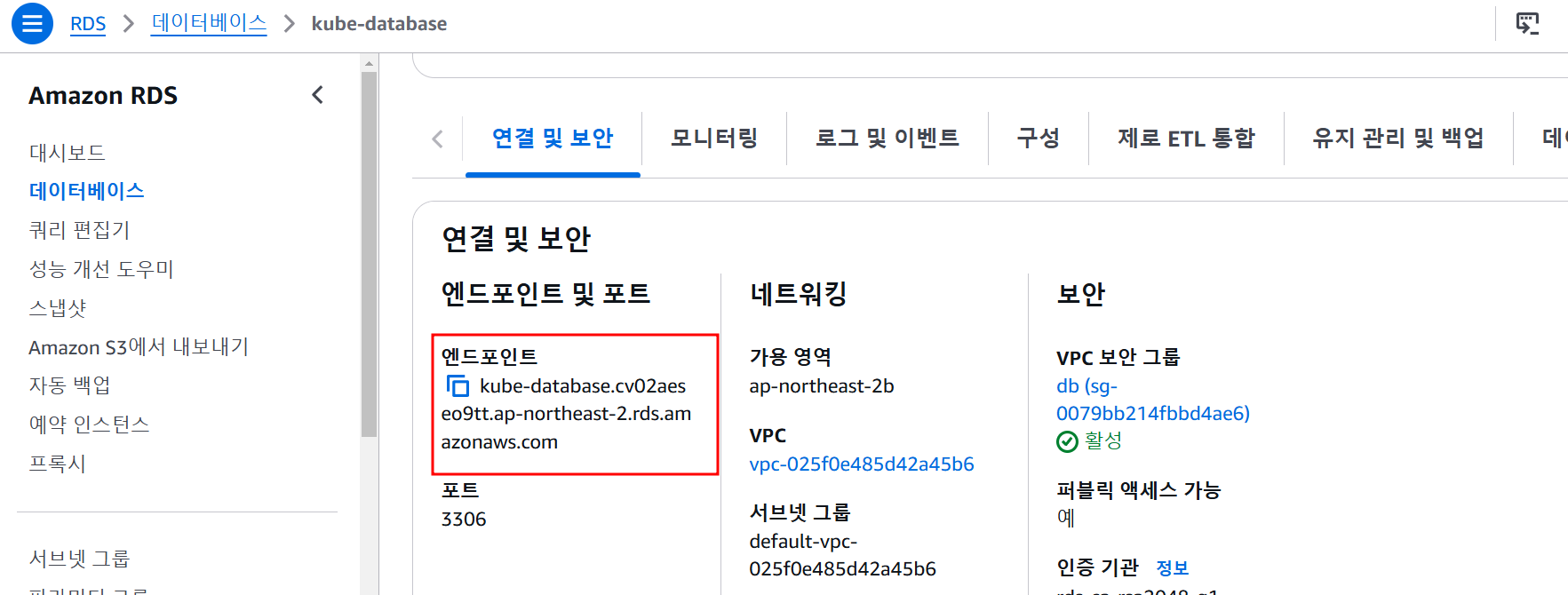
| apiVersion: apps/v1  kind: Deployment  # Deployment 기본 정보  metadata:  name: spring-deployment # Deployment 이름  # Deployment 세부 정보  spec:  replicas: 3 # 생성할 파드의 복제본 개수  selector:  matchLabels:  app: backend-app # 아래에서 정의한 Pod 중 'app: backend-app'이라는 값을 가진 파드를 선택  # 배포할 Pod 정의  template:  metadata:  labels: # 레이블 (= 카테고리)  app: backend-app  spec:  **imagePullSecrets:**  **- name: regcred**  containers:  - name: spring-container # 컨테이너 이름  # ECR에 저장된 이미지의 주소 및 태그명  image: **002177417362.dkr.ecr.ap-northeast-2.amazonaws.com/kube-ecr:1.0**  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 |
| --- |

* + imagePullSecrets : 이미지를 Pull 할 때 해당 Secret 값을 활용해서 Private ECR 레지스트리에 접근한다.



spring-config.yaml

| apiVersion: v1  kind: ConfigMap  # ConfigMap 기본 정보  metadata:  name: spring-config # ConfigMap 이름  # Key, Value 형식으로 설정값 저장  data:  # RDS 엔드포인트 작성하기  db-host: **kube-database.coseefawhrzc.ap-northeast-2.rds.amazonaws.com**  db-port: "3306"  db-name: mydb |
| --- |

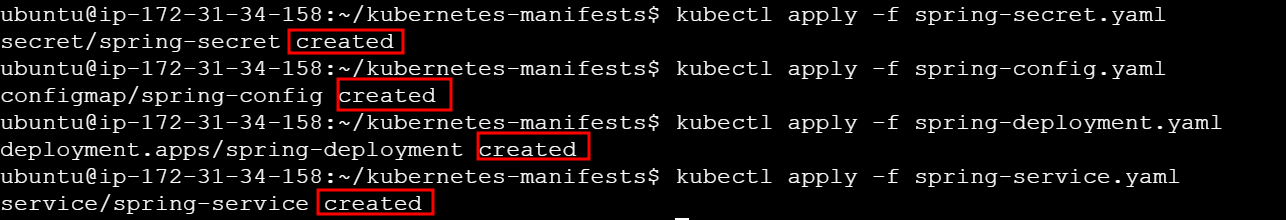


spring-secret.yaml

| apiVersion: v1  kind: Secret  type: Opaque # 임의의 사용자 정의 데이터를 저장할 때 사용하는 타입  # Secret 기본 정보  metadata:  name: spring-secret # Secret 이름  # Key, Value 형식으로 값 저장  stringData:  # RDS 생성 시 기재한 **사용자 이름**과 **마스터 비밀번호** 입력하기  db-username: **admin**  db-password: **password** |
| --- |

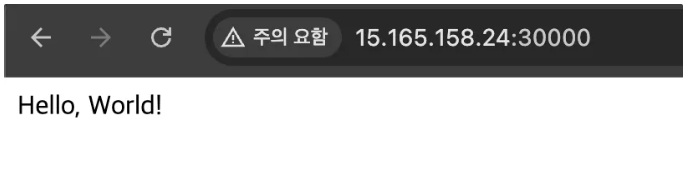
1. **매니페스트 파일을 통해 오브젝트 생성하기**

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

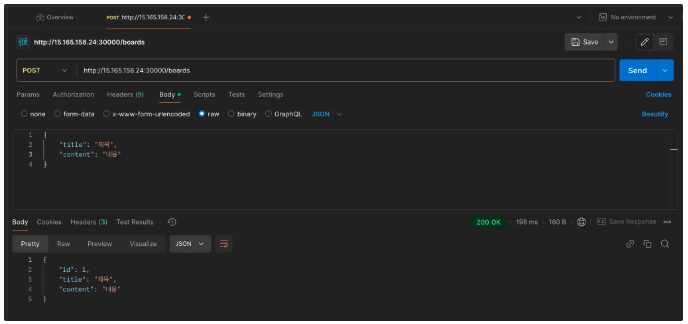
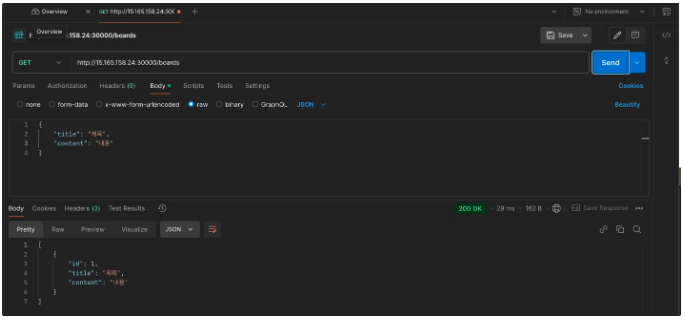


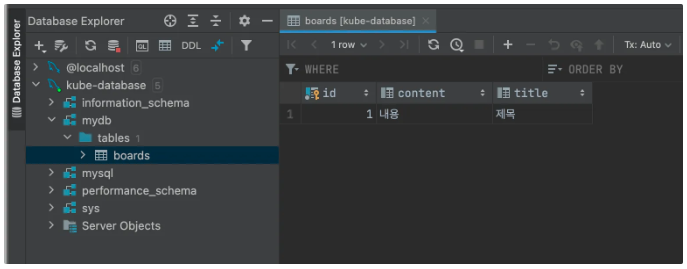
1. **잘 생성 됐는지 확인**

| $ kubectl get secret  $ kubectl get configmap  $ kubectl get deployment  $ kubectl get service  $ kubectl get pods |
| --- |

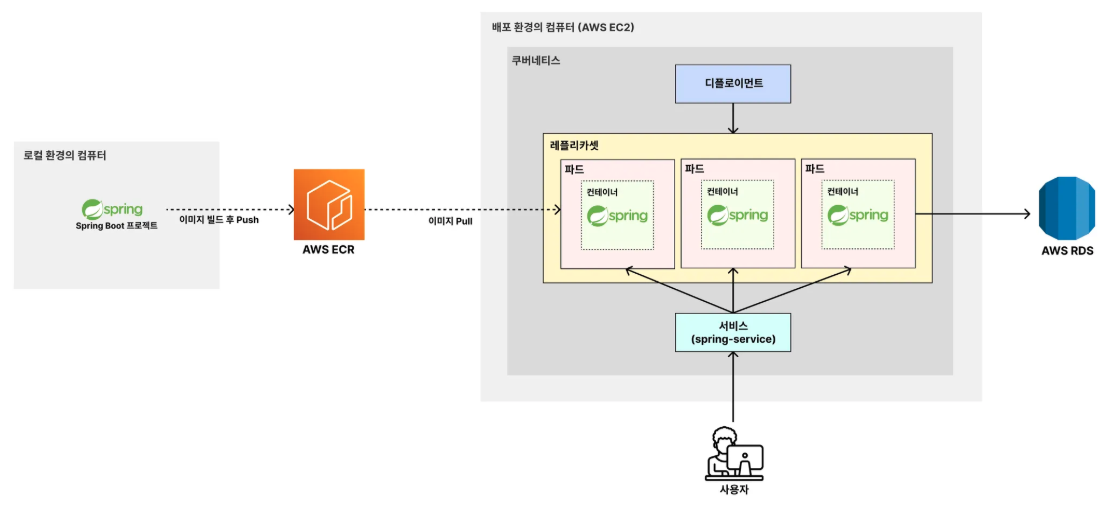
1. **API가 잘 작동하는 지 확인**  
   

포스트맨 검색 다운받아서 설치하기

1. **DB에 정상적으로 데이터가 저장되는 지 확인**  
   

### **✅ 아키텍처 다시 한 번 짚어보기**



1. **EC2에 배포된 백엔드(Spring Boot) 서버 업데이트하기**

### **✅ EC2에 배포된 백엔드(Spring Boot) 서버 업데이트하기**

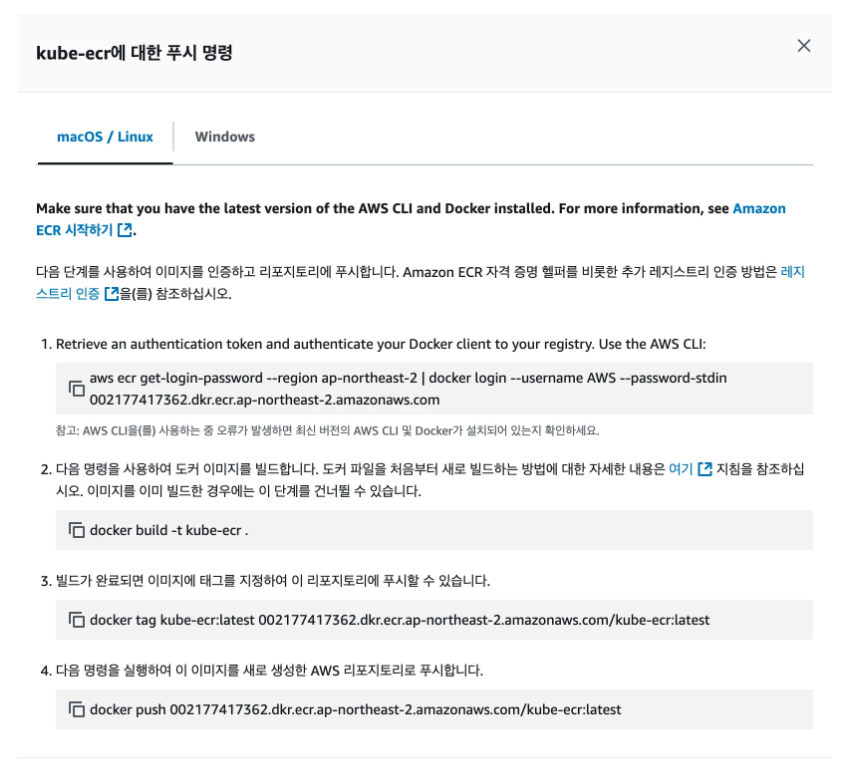
1. **로컬 환경에서 Spring Boot 코드 수정하기**

AppController

| @RestController  public class AppController {  @GetMapping("/")  public String home() {  **return "Version 2.0";**  }  } |
| --- |

1. **빌드해서 ECR로 이미지 업로드하기**

| $ ./gradlew clean build  $ aws ecr get-login-password --region ap-northeast-2 | docker login --username AWS --password-stdin 002177417362.dkr.ecr.ap-northeast-2.amazonaws.com  $ docker build -t kube-ecr .  $ docker tag kube-ecr:latest 002177417362.dkr.ecr.ap-northeast-2.amazonaws.com/kube-ecr:**2.0**  $ docker push 002177417362.dkr.ecr.ap-northeast-2.amazonaws.com/kube-ecr:**2.0** |
| --- |

****

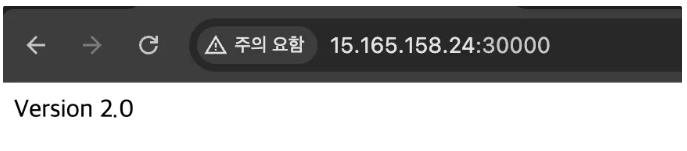
1. **EC2에 있는 매니페스트 파일 수정하기**

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:  imagePullSecrets:  - name: regcred  containers:  - name: spring-container # 컨테이너 이름  image: 002177417362.dkr.ecr.ap-northeast-2.amazonaws.com/kube-ecr:**2.0** # 컨테이너를 생성할 때 사용할 이미지  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. **수정된 매니페스트 파일을 기반으로 업데이트하기**

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

1. **업데이트 됐는 지 확인하기**  
   
2. **남들보다 훨씬 쿠버네티스를 빠르게 배우는 방법?!**

### **✅ 쿠버네티스에 빠르게 익숙해지려면?!**

프로그래밍에서 어떤 기술을 빠르게 익히려면 많이 써봐야 한다. 불변의 법칙이다. 어떤 기술이든 한 번만에 잘 쓰는 방법 따위는 없다. 이렇게도 써보고, 저렇게도 써보고, 장애도 내보고, 응용도 해봐야만 그 기술이 익숙해진다.

프로그래밍 언어를 빠르게 배우는 방법에서 프로젝트를 하라는 말을 한 번쯤 들어봤을 것이다. 프로젝트를 해야만 이론들을 실전에서 써먹을 수 있게 되고, 수많은 시행착오를 만나면서 익숙해지게 된다.

쿠버네티스도 똑같다. 개인 프로젝트에서 쿠버네티스를 활용해서 서비스를 운영해보는 경험을 해봐야 많이 배울 수 있다. 당연히 개인 프로젝트나 작은 규모의 기업에서 쿠버네티스를 쓰는 게 오버스펙인 건 안다. 하지만 우리는 쿠버네티스 자체를 학습하는 것에 목표를 뒀다면, 쿠버네티스를 많이 쓸 수 밖에 없는 환경을 만들어야 한다. 즉, 자주 써먹을 수 있고 마음대로 가지고 놀 수 있는 환경을 만드는 게 중요하다.

지금까지 배운 EC2에 k3s를 활용하는 방식을 활용한다면 비용을 크게 들이지 않고 마음껏 쿠버네티스를 연습할 수 있게 된다. 따라서 이번에 배운 내용을 반드시 실제 프로젝트에 적용시키면서 연습해봐라. 쿠버네티스를 활용해 배포도 해보고 운영도 해봐라. 그러면서 다양한 경험을 쌓아나가라.