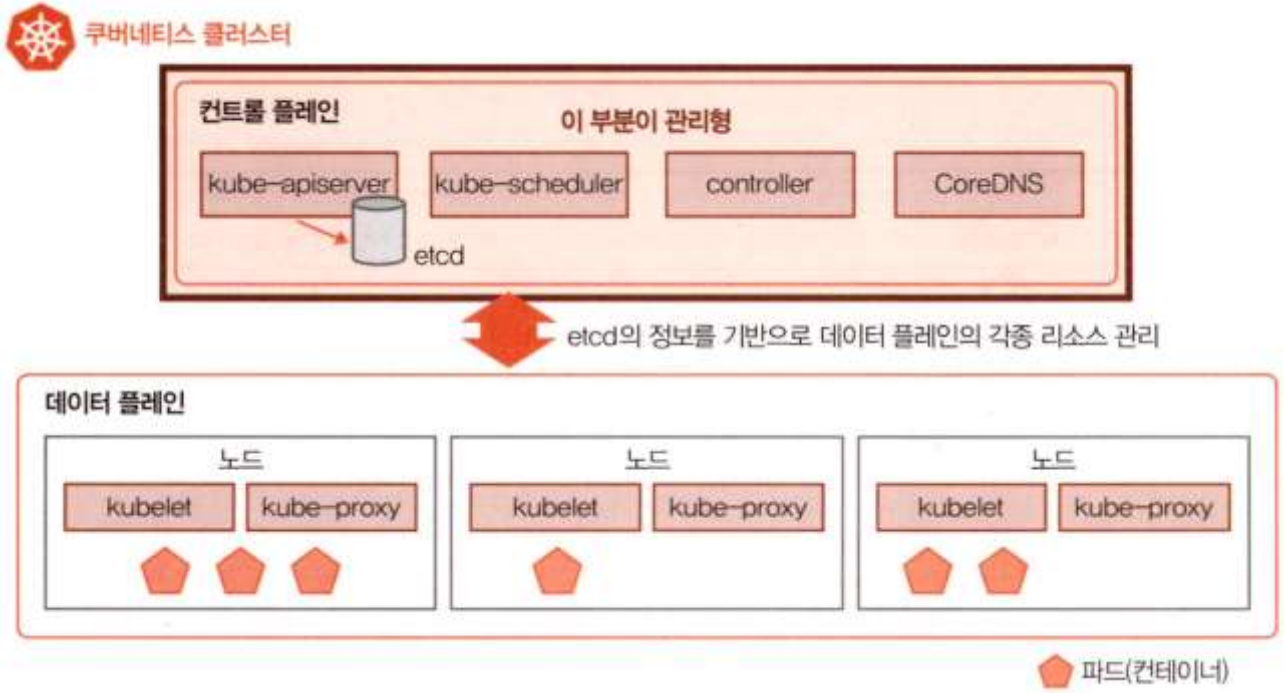
EKS

개요

❖ EKS 개요

- ✓ Amazon EKS(Elastic Kubernetes Service)는 Kubernetes를 제어하는 Control Plain을 제공하는 관리형 서비스
- ✓ AWS가 주최하는 최대 연례 행사인 re:Invent 2017에서 발표되었으며 도쿄 region에는 2018년 12월 서울 region에는 2019년 1월에 출시

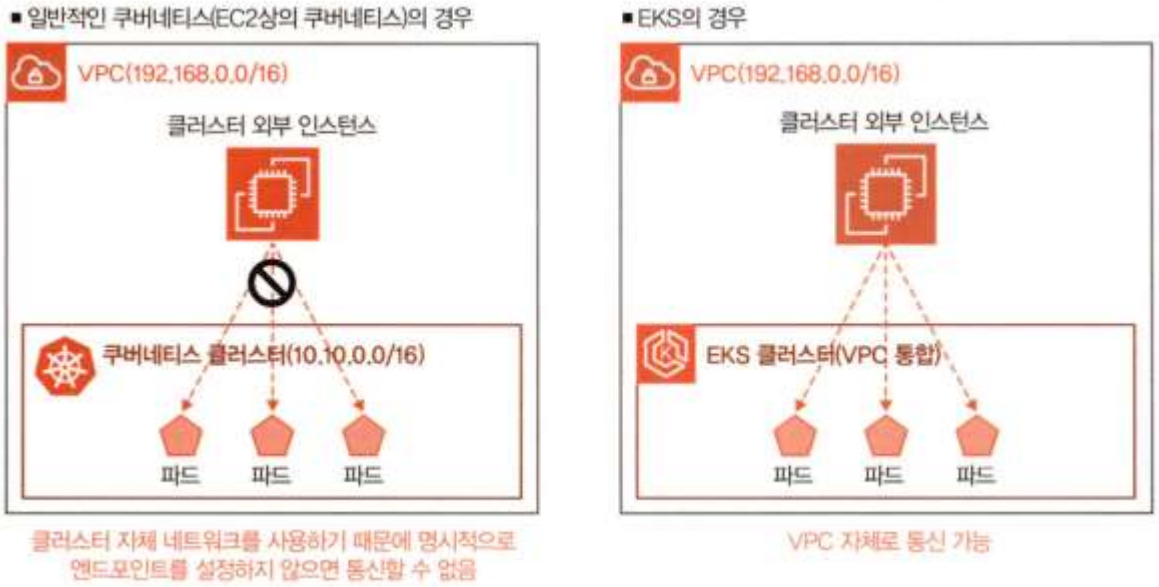


개요

❖ EKS 특징

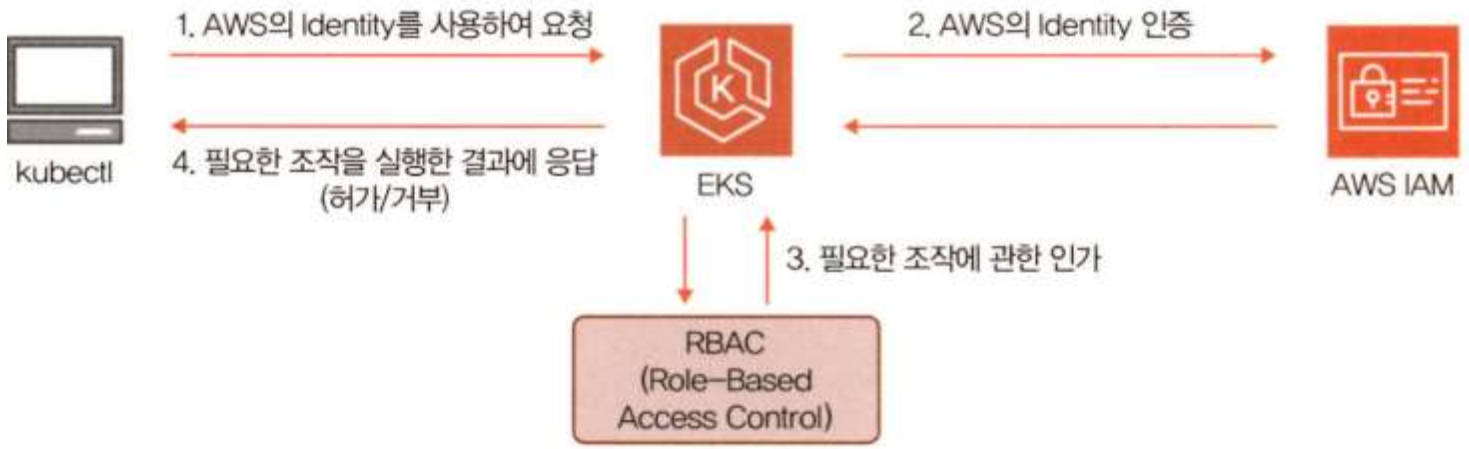
✓ VPC 와 통합

- Kubernetes Cluster에서는 Pod Network로 Data Network의 Network 와는 다른 자체 네트워크 체계를 배치하기 때문에 클러스터 외부에서 Pod에 명시적으로 End Point를 생성하지 않으면 통신이 불가능
- EKS에서는 Amazon VPC 통합 네트워킹을 지원하고 있어 Pod에서 VPC 내부 주소 대역을 사용할 수 있고 클러스터 외부와의 통신을 Seamless(서비스 접근을 단순하게 하는 것 또는 복잡한 기술이나 기능을 설명하지 않아도 서비스 기능을 직관적으로 구현하는 것) 하게 구현할 수 있음



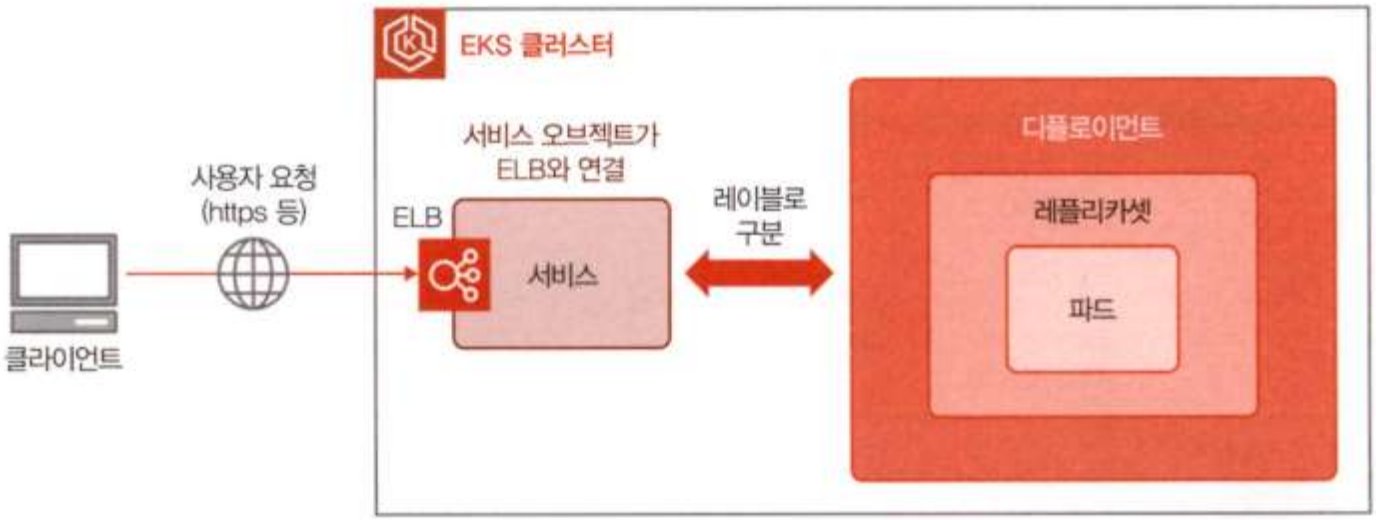
개요

- ❖ EKS 특징
  - ✓ IAM을 통한 인증 과 인가
    - kubernetes 클러스터는 kubectl 이라는 명령 도구를 사용하여 조작하는데 해당 조작이 허가된 사용자에 의한 것임을 올바르게 인증해야 하고 인증된 사용자에게 어떤 조작을 허가 할지에 대한 인가 구조도 필요
    - AWS 사용자라면 어떤 시스템을 구축할 경우 IAM을 사용하고 있기 때문에 여기서 관리하는 IAM 사용자나 IAM 역할을 사용해 kubernetes 클러스터의 인증 및 인가를 할 수 있으므로 편리



개요

- ❖ EKS 특징
  - ✓ ELB 와 연계
    - kubernetes 클러스터 외부에서 접속할 때는 서비스를 사용해 End Point를 생성할 필요가 있는데 가장 전형적인 End Point가 Load Balancer
    - EKS에서는 kubernetes의 서비스 타입 중 하나인 Load Balancer를 설정하면 자동적으로 Load Balancer 서비스인 ELB가 생성되는데 이것으로 HTTPS나 경로 기반 라우팅 등의 L7 Load Balancer 기능을 AWS 서비스로 구현할 수 있음



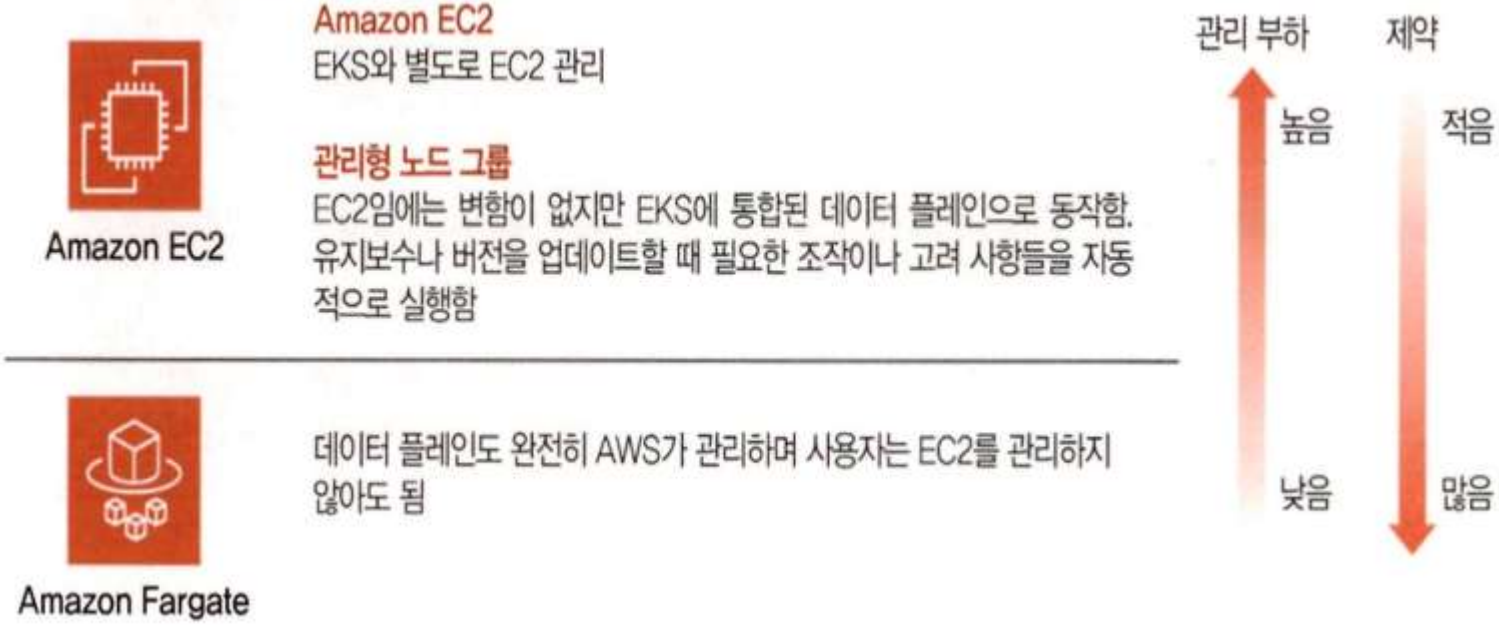
개요

- ❖ EKS 특징
  - ✓ 데이터 플레인 선택
    - Kubernetes는 컨트롤 플레인과 데이터 플레인으로 구성되는데 컨트롤 플레인은 EKS에서 관리형 서비스로 제공
    - EKS 서비스 제공 초기에는 AWS가 자동화된 구축 방식으로 데이터 플레인을 제공했고 EKS 와는 별도로 EC2를 관리해야 했지만 시간이 지나면서 AWS가 발전해 데이터 플레인 관리를 도와주는 기능이 제공됨
    - EKS 클러스터의 유지 관리나 버전을 업그레이드할 때 필요한 가상 머신 설정을 쉽게 해주는 관리형 노드 그룹 구조와 처음부터 가상 머신을 의식하지 않고 Pod를 배포할 수 있는 Fargate 라는 서비스를 제공했지만 관리형 노드 그룹에서는 AWS에서 제공한 것만 기본 이미지로 사용할 수 있었는데 기업의 표준 보안 소프트웨어를 설치하거나 노드에 다른 도구를 설치한 사용자 컨테이너 이미지 사용 등이 불가능했는데 이러한 문제를 해결하고자 2020년 8월 노드 그룹에서 사용자 설정 컨테이너 이미지를 지원하는 기능이 출시되어 사용자가 설정한 컨테이너 이미지로 노드를 생성할 수 있게 됨
    - Fargate는 데이터 플레인 관리가 필요 없는 만큼 Pod가 배포되는 호스트에 사용자 접속도 제한되며 거기에 따른 제약도 있음

개요

❖ EKS 특징

✓ 데이터 플레인 선택



쿠버네티스 환경 구축 과 배포

- EKS 구축에 사용하는 도구
  - ✓ EKS 클러스터를 구축하는 방법
    - EKS 클러스터 구축 도구 eksctl을 이용
    - AWS 관리 콘솔 또는 AWS CLI을 이용
  - ✓ eksctl
    - eksctl은 EKS 클러스터 구축 및 관리를 하기 위한 오픈소스 명령 도구
    - 기본 구성이라면 eksctl create cluster 란 명령만으로 EKS 클러스터를 구축할 수 있음
    - 다양한 옵션을 설정함으로써 유연하게 구성을 변경할 수 있음
    - 활발한 오픈소스 프로젝트이기 때문에 계속 기능이 추가되고 있기도 있음
    - eksctl을 사용하면 VPC, Subnet, 보안 그룹 등 EKS 클러스터를 구축하는 데 필요한 리소스를 한번에 구성할 수 있음
    - EKS 클러스터는 정지 해도 과금되기 때문에 이를 막으려면 클러스터를 삭제해야 함
    - eksctl로 VPC 등을 생성한 경우 EKS 클러스터를 삭제할 때 VPC 등도 함께 삭제되어 버림
    - EKS 클러스터와 VPC 등의 리소스를 별도로 구축하면 EKS 클러스터를 삭제해도 다른 리소스가 삭제되지 않으므로 EKS 클러스터만 재생성하면 다시 작업을 시작할 수 있음
    - EC2나 RDS와 같이 정지 상태가 되면 과금되지 않는 리소스도 있지만 VPC를 삭제하면 이런 리소스도 먼저 삭제해야 하므로 작업을 재개할 때 효율이 떨어짐

쿠버네티스 환경 구축 과 배포

- EKS 구축에 사용하는 도구
  - ✓ VPC 관련 용어
    - AWS에서는 서버 등의 리소스를 VPC라는 논리적으로 분리된 영역을 이용하여 관리하는데 1개의 AWS 계정에 VPC 여러 개를 생성할 수 있지만 기본적으로 VPC 끼리는 독립적인 환경이고 명시적으로 VPC를 연결(피어링)하지 않는 한 VPC 간 통신은 할 수 없음
    - AWS 데이터센터는 전 세계에 존재하지만 모두 리전이라고 하는 지리적 영역에 속한 형태인데 대부분의 AWS 서비스는 모두 리전을 선택해 리소스를 생성하게 되어 있음
    - VPC는 리전 여러 개를 선택해서 생성할 수 없기 때문에 기본적으로 1개의 리전을 선택하여 그 안에 서비스를 구축
    - 재해 시의 백업 등을 고려하여 여러 리전에 리소스를 배치하는 경우도 있지만 서비스 제공에 필요한 리소스는 하나의 리전에 배치하고 백업용 스토리지 등을 다른 리전에 배치하는 형태가 일반적
    - 리전 안에는 가용 영역이 여러 개 있는데 가용 영역들은 리전 내에서 물리적으로 떨어진 장소에 존재하므로 가용 영역 여러 개에 리소스를 배포하여 다중화해두면 데이터센터 수준의 장애가 발생해도 서비스를 계속할 수 있음
    - VPC는 서브넷을 사용해 네트워크를 분할하여 관리
    - 서브넷은 가용 영역 여러 개를 동시에 사용할 수 없으며 가용 영역 여러 개를 사용할 경우 서브넷도 그만큼 나눠야 함
    - 라우팅 설정은 서브넷 단위로 할 수 있기 때문에 인터넷에 직접 접속할 수 있는 서브넷(퍼블릭 서브넷) 과 직접 접속할 수 없는 서브넷(프라이빗 서브넷)을 생성할 수 있음

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구
  - ✓ CloudFormation 을 이용한 환경 구축
    - AWS에 리소스를 구축할 때 AWS에서 제공하는 환경 구축 도구로는 CloudFormation 과 HashiCorp 사의 Terraform을 주로 사용
    - AWS 리소스는 AWS 관리 콘솔(AWS가 제공하는 웹 UI) 이나 AWS CLI 등 여러 방법으로 구축 가능하지만 관련 리소스를 한번에 구축하고 필요에 따라 변경 및 삭제할 때 CloudFormation 적합
    - CloudFormation에서는 JSON 또는 YAML 형식으로 리소스 구성을 정의하고 실행은 AWS 관리 콘솔, AWS CLI(AWS가 제공하는 AWS 리소스를 관리하기 위한 명령줄 도구) 에서 모두 가능

쿠버네티스 환경 구축 과 배포

- EKS 구축에 사용하는 도구
  - ✓ AWS 관리 콘솔
    - AWS 관리 콘솔은 AWS 서비스를 관리하기 위한 웹 사용자 인터페이스
    - AWS의 가장 일반적인 조작 방법이며 GUI 기반이므로 AWS에 익숙하지 않은 경우에도 직관적으로 쉽게 사용할 수 있음
    - AWS 리소스 환경 구축을 주로 CloudFormation 으로 하지만 CloudFormation 자체는 AWS 관리 콘솔을 이용해 실행하며 CloudFormation으로 구축한 환경 확인도 AWS 관리 콘솔을 이용
  - ✓ AWS CLI
    - AWS가 제공하는 명령줄 도구로 AWS 관리 콘솔과 함께 AWS 리소스 관리에 많이 사용
    - AWS CLI는 파이썬으로 만들어진 도구로 인스톨러가 제공되며 파이썬 패키지 관리 도구인 pip를 이용해 설치할 수도 있음

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ Access Key ID 와 Secret Access Key 생성
    - ● 사용자를 생성하고 권한 추가
      - ◆ AmazonEKSClusterPolicy
      - ◆ AmazonEKSServicePolicy
      - ◆ AmazonEKSWorkerNodePolicy
      - ◆ AmazonVPCFullAccess
      - ◆ IAMFullAccess

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ Access Key ID 와 Secret Access Key 생성
    - ● 생성한 사용자에게 인라인 정책을 이용해서 권한 추가

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*",
                "ec2:Describe*",
                "ec2:CreateSecurityGroup",
                "ec2:DeleteSecurityGroup",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:AuthorizeSecurityGroupEgress",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:PutRolePolicy",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:CreateInstanceProfile",
                "iam:AddRoleToInstanceProfile",
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ Access Key ID 와 Secret Access Key 생성
    - ● 생성한 사용자에게 인라인 정책을 이용해서 권한 추가

```
"iam:RemoveRoleFromInstanceProfile",
"iam:DeleteInstanceProfile",
"cloudformation:*",
"autoscaling:DescribeAutoScalingGroups",
"autoscaling:CreateAutoScalingGroup",
"autoscaling:DeleteAutoScalingGroup",
"autoscaling:UpdateAutoScalingGroup",
"ec2:CreateLaunchTemplate",
"ec2:DeleteLaunchTemplate",
"ec2:DescribeLaunchTemplates",
"ec2:DescribeLaunchTemplateVersions",
"ec2:CreateTags",
"ec2:RunInstances",
"ec2:DescribeInstances",
"ec2:TerminateInstances",
"ec2:AttachNetworkInterface",
"ec2:CreateNetworkInterface",
"ec2:DeleteNetworkInterface",
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ Access Key ID 와 Secret Access Key 생성
    - ● 생성한 사용자에게 인라인 정책을 이용해서 권한 추가

```
                "iam:PassRole",
                "iam:CreateServiceLinkedRole",
                "iam:GetServiceLinkedRole",
                "iam:ListAttachedRolePolicies",
                "iam:ListInstanceProfiles",
                "iam:ListRoles"
            ],
            "Resource": "*"
        }
    ]
}
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ Access Key ID 와 Secret Access Key 생성
    - ● 생성한 사용자의 보안 자격 증명 탭에서 액세스 키 만들기

쿠버네티스 환경 구축 과 배포

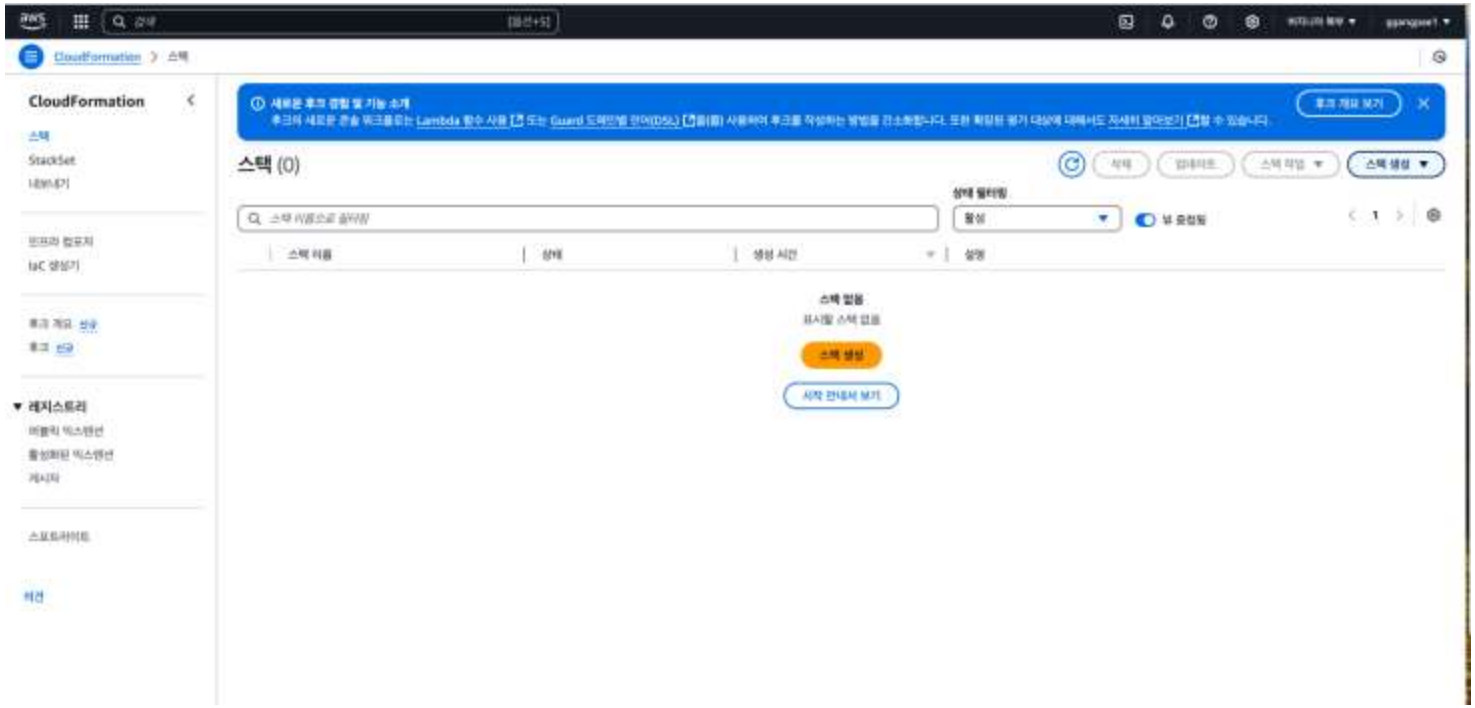
- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ AWS CLI
    - 다운로드: https://docs.aws.amazon.com/ko_kr/cli/latest/userguide/getting-started-install.html
    - 확인: aws --version
    - aws configure 명령을 수행해서 access-key 와 secret access-key를 설정

쿠버네티스 환경 구축 과 배포

- ❖ EKS 구축에 사용하는 도구 설치
  - ✓ eksctl 설치
    - Windows: https://github.com/eksctl-io/eksctl/releases
    - Mac: brew install weaveworks/tap/eksctl
    - 설치 확인: eksctl version
  - ✓ kubectl
    - Windows: https://kubernetes.io/ko/docs/tasks/tools/install-kubectl-windows/
    - Mac: brew install kubectl
    - Ubuntu
      - ◆ curl -LO "https://dl.k8s.io/release/v1.30.0/bin/linux/amd64/kubectl"
      - ◆ chmod +x ./kubectl
      - ◆ sudo mv ./kubectl /usr/local/bin/kubectl
    - 설치 확인: kubectl version --client

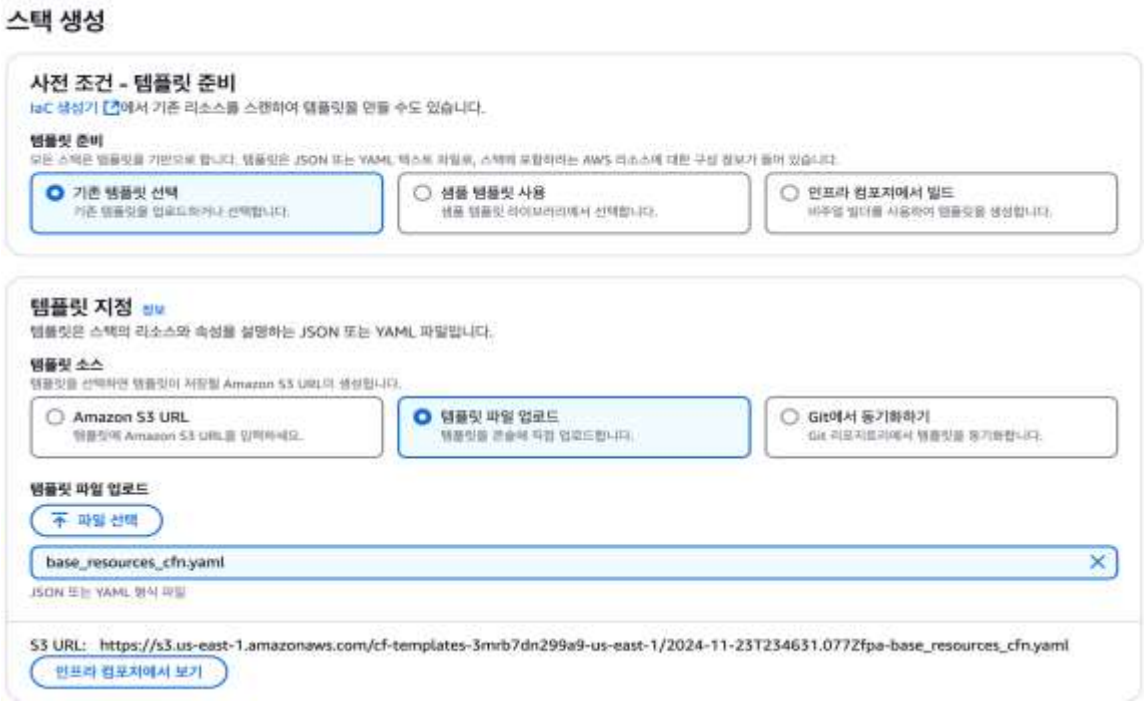
쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 기본 리소스 생성
      - ◆ CloudFormation 페이지로 이동
      - ◆ [스택(CloudFormation 으로 생성하는 리소스) 생성] 버튼을 클릭



쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 기본 리소스 생성
      - ◆ [사전 조건]에서는 [기존 템플릿 선택]을 선택
      - ◆ [템플릿 지정]에서 [템플릿 파일 업로드]를 선택한 후 [base_resource_cfn.yaml] 파일을 선택

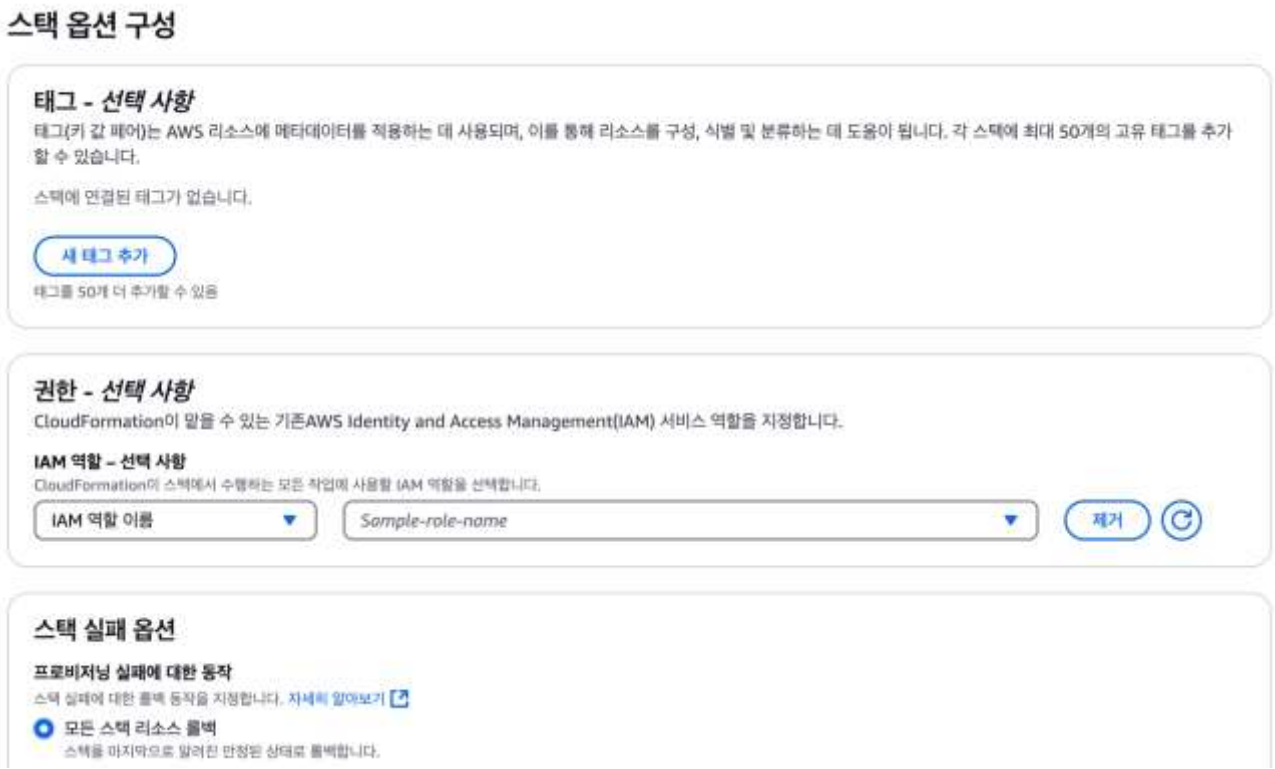


쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 기본 리소스 생성
      - ◆ 스택 세부 정보 지정 페이지에서 스택 이름을 설정(eks-work-base)

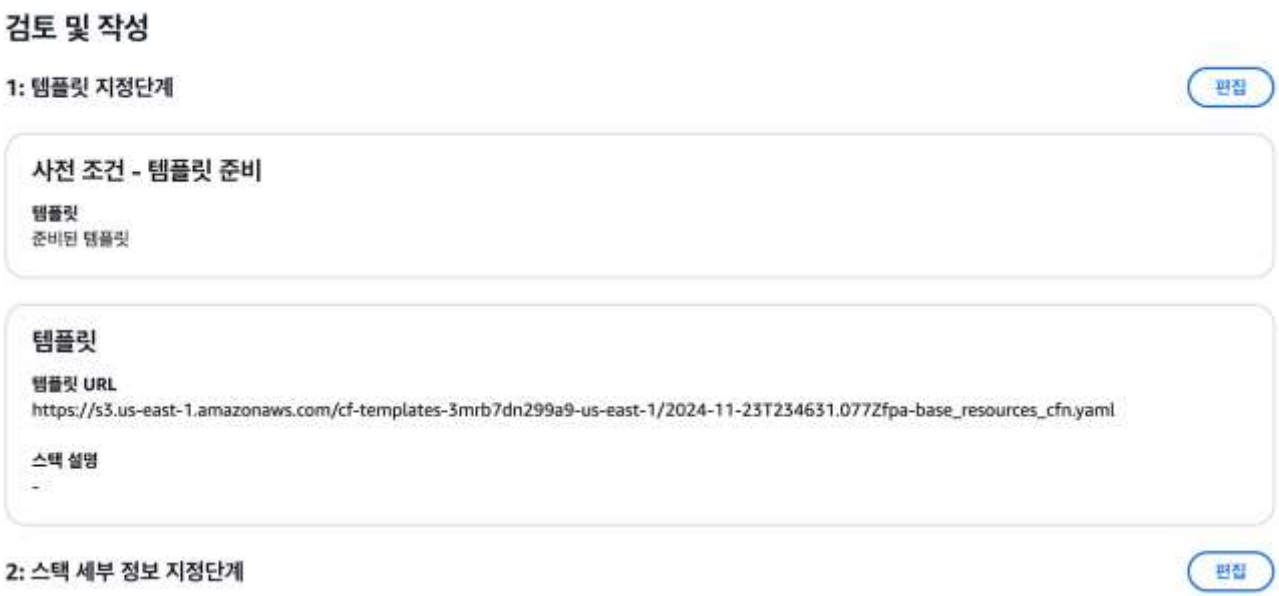
쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 기본 리소스 생성
      - ◆ 스택 옵션 구성 페이지는 설정 변경없이 다음 버튼 클릭



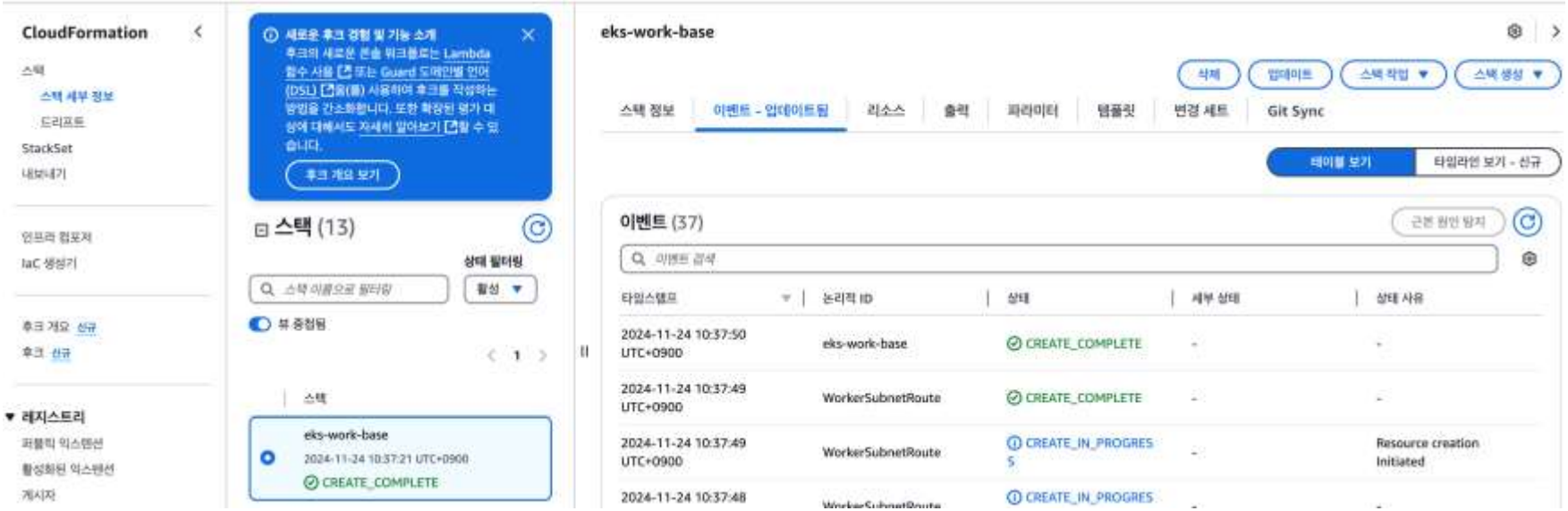
쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 기본 리소스 생성
      - ◆ 설정 변경 없이 [전송] 버튼 클릭



쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 기본 리소스 생성

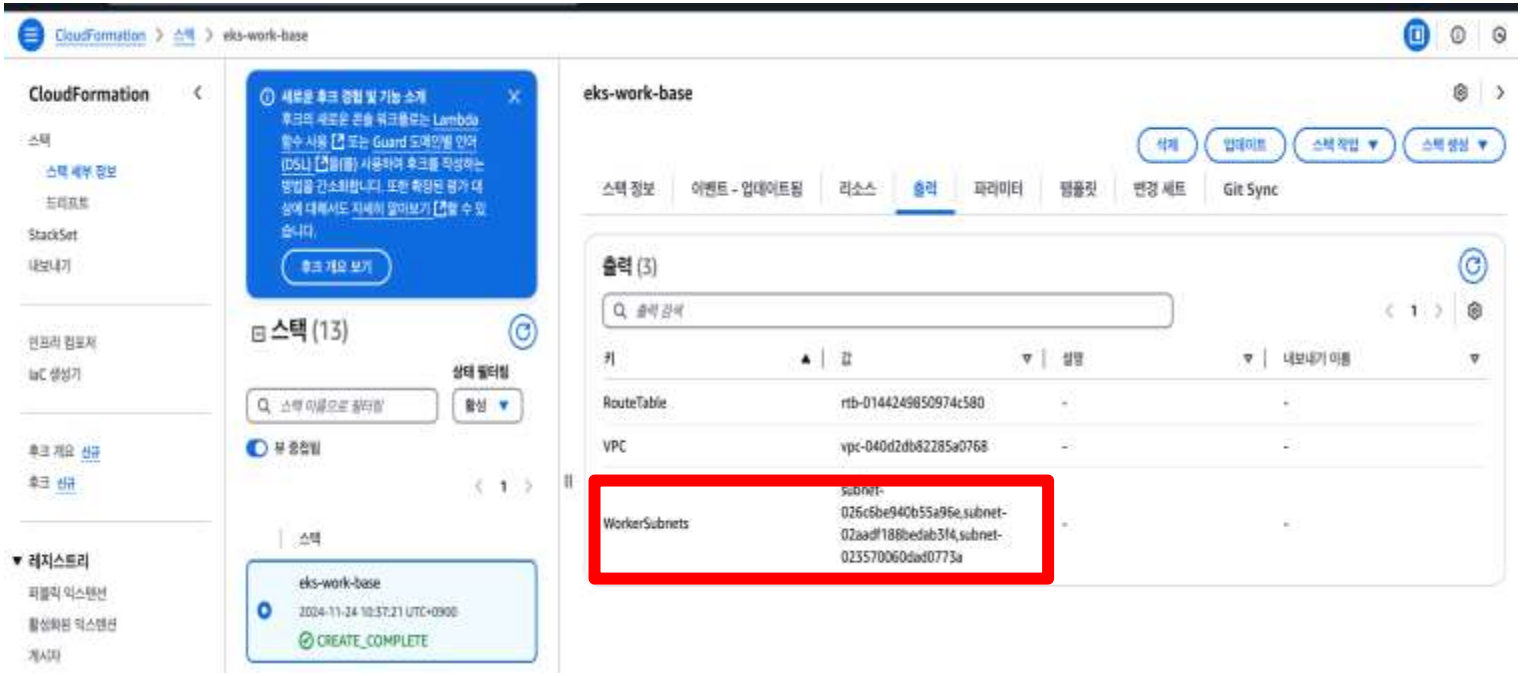


쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ 기본 리소스 구축
    - ● 생성된 리소스 확인: VPC 항목에서 확인

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● 기본 리소스 정보를 확인
      - ◆ 출력 탭에서 WorkerSubnets 값 복사



쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● eksctl 실행: 시간이 소요됨

```
$ eksctl create cluster \                          # 아래의 인수를 설정하여 eksctl 실행
> --vpc-public-subnets <WorkerSubnets값> \         # 워커 노드용 서브넷
> --name eks-work-cluster \                        # 클러스터 이름
> --region ap-northeast-2 \                        # 리전(서울 리전을 설정)
> --version 1.19 \                                 # EKS 클러스터 버전
> --nodegroup-name eks-work-nodegroup \            # 노드 그룹 이름
> --node-type t2.small \                           # 워커 노드 인스턴스 타입
> --nodes 2 \                                      # 워커 노드 수
> --nodes-min 2 \                                  # 워커 노드의 최소 노드 수
> --nodes-max 5                                    # 워커 노드의 최대 노드 수
```

eksctl create cluster --vpc-public-subnets subnet-00a45c540767d387f,subnet-013cfcba73b595d53,subnet-07b4b3cbb97bbe13d --name eks-work-cluster --region ap-northeast-2 --version 1.30 --nodegroup-name eks-work-nodegroup --node-type t2.small --nodes 2 --nodes-min 2 --nodes-max 5

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● eksctl 실행
      - ◆ 실행 결과: CloudFormation에서 확인하면 스택이 2개 추가됨

eksctl-eks-work-cluster-nodegroup-eks-work-nodegroup	CREATE_COMPLETE	2024-11-25 08:02:48 UTC+0900	EKS Managed Nodes (SSH access: false) [created by eksctl]
eksctl-eks-work-cluster-cluster	CREATE_COMPLETE	2024-11-25 07:52:44 UTC+0900	EKS cluster (dedicated VPC: false, dedicated IAM: true) [created and managed by eksctl]
eks-work-base	CREATE_COMPLETE	2024-11-24 10:37:21 UTC+0900	-

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● eksctl 실행
      - ◆ 실행 결과: EC2 인스턴스가 워커 노드의 개수만큼 추가됨

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● kubeconfig 설정
      - ◆ eksctl은 EKS 클러스터 구축 중에 kubeconfig(kubectl 을 설치하면 생성되는 파일인 ~/.kube/config) 파일을 자동으로 업데이트
      - ◆ .kubeconfig 파일은 쿠버네티스 클라이언트인 kubectl이 이용할 설정 파일로 접속 대상 쿠버네티스 클러스터의 접속 정보(컨트롤 플레인 URL, 인증 정보, 이용할 쿠버네티스의 네임스페이스 등)를 저장하고 있음
      - ◆ EKS 클러스터에 접속하기 위한 인증 정보는 AWS CLI로 확인 가능하며 eksctl을 사용하면 AWS CLI를 호출하여 인증하기 위한 설정을 kubeconfig 파일에 포함할 수 있음
      - ◆ EKS 인증은 AWS CLI에서 가능하며 eksctl에서도 이것을 이용하도록 설정할 수 있는데 AWS CLI 자체에도 EKS 클러스터를 지정하고 kubeconfig 파일을 업데이트할 수 있는 기능이 있음

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● kubeconfig 설정
      - ◆ kubeconfig 구성
        - ▪ clusters: 쿠버네티스 클러스터의 정보
        - ▪ users
          - 클러스터에 접근할 유저의 정보
          - 각 환경마다 필요한 값들이 다름
        - ▪ context: cluster와 user를 조합해서 생성된 값
        - ▪ current-context: 현재 사용하는 context 를 지정하는 부분

쿠버네티스 환경 구축 과 배포

❖ EKS 클러스터 구축

✓ EKS 클러스터 구축

- kubeconfig 설정
  - ◆ context 조회: kubectl config get-contexts
    - ▪ 행 앞에 '*' 표시가 있는 행이 현재 활성화된 컨텍스트

```
CURRENT   NAME                                                  CLUSTER                                        AUTHINFO
                              NAMESPACE
          docker-desktop                                        docker-desktop                                 docker-desktop

*         iam-root-account@eks-work-cluster.ap-northeast-2.eksctl.io   eks-work-cluster.ap-northeast-2.eksctl.io   iam-root-account@eks-
work-cluster.ap-northeast-2.eksctl.io
```

  - ◆ context 변경: kubectl config use-context 컨텍스트이름

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● kubeconfig 설정
      - ◆ 노드 상태 조회: kubectl get nodes

```
[redacted] kubectl get nodes
NAME                                              STATUS   ROLES    AGE   VERSION
ip-192-168-0-199.ap-northeast-2.compute.internal  Ready    <none>   23h   v1.31.2-eks-94953ac
ip-192-168-2-237.ap-northeast-2.compute.internal  Ready    <none>   23h   v1.31.2-eks-94953ac
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● EKS 클러스터 동작 확인
      - ◆ Pod 배포를 위한 yaml 파일 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx-app
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● EKS 클러스터 동작 확인
      - ◆ kubectl apply –f 야믈파일경로
      - ◆ kubectl get pods

```
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          2m50s
```

      - ◆ kubectl port-forward nginx-pod 8080:80

```
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

      - ◆ 브라우저에서 localhost:8080 으로 접속

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● EKS 클러스터 동작 확인
      - ◆ CTRL + C를 눌러서 port-forwarding 종료

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● EKS 클러스터 동작 확인
      - ◆ 서비스를 위한 yaml 파일 생성

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: LoadBalancer
  selector:
    app: nginx-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● EKS 클러스터 동작 확인
      - ◆ 서비스를 위한 yaml 파일 실행
      - ◆ 리소스 확인: kubectl get all
      - ◆ 서비스의 external-IP로 브라우저에서 접속

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터 구축
  - ✓ EKS 클러스터 구축
    - ● EKS 클러스터 삭제
      - ◆ 클러스터에서 실행 중인 모든 서비스를 나열

        kubectl get svc --all-namespaces
      - ◆ EXTERNAL-IP 값과 연결된 모든 서비스를 삭제해야 하는데 이를 이용해서 Elastic Load Balancer를 삭제

        kubectl delete svc service-name
      - ◆ 클러스터 및 연결된 노드를 삭제하되 prod를 클러스터 이름으로 변경

        eksctl delete cluster --name prod

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 데이터베이스 환경 설정
    - Amazon RDS로 오픈소스 관계형 데이터베이스인 PostgreSQL을 구축하여 사용
    - 서비스 환경에서 사용할 경우 데이터베이스는 가용 영역 여러 개로 다중화하여 구축하는 것이 일반적
    - Amazon RDS는 AWS가 제공하는 관계형 데이터베이스의 관리형 서비스로 RDS를 사용하면 AWS 관리 콘솔이나 AWS CLI, ClouDFormation 등을 이용해 데이터베이스를 구축할 수 있을 뿐만 아니라 통상적인 데이터베이스 운영에 관련된 작업 부하를 줄일 수 있음
    - Amazon RDS에는 데이터베이스가 설치된 OS에 로그인할 수 없는 등 데이터베이스 환경 관련 제약 사항이 존재
    - 데이터베이스 관리용 서버로 Bastion Host를 구축하여 사용
      - ◆ Bastion Host란 외부에서 내부 네트워크에 접근할 수 있는 유일한 방법인 접근점
      - ◆ 보안성이 높은 인프라와 외부 인터넷을 연결하는 중계 서버로 작동하며 모든 인바운드 트래픽은 Bastion Host를 통과해야 내부 네트워크로 들어갈 수 있음
    - Bastion Host는 AWS 가상 서버인 EC2로 구축

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 데이터베이스 환경 구축
    - Amazon RDS로 오픈소스 관계형 데이터베이스인 PostgreSQL을 구축하여 사용
    - 서비스 환경에서 사용할 경우 데이터베이스는 가용 영역 여러 개로 다중화하여 구축하는 것이 일반적
    - Amazon RDS는 AWS가 제공하는 관계형 데이터베이스의 관리형 서비스로 RDS를 사용하면 AWS 관리 콘솔이나 AWS CLI, ClouDFormation 등을 이용해 데이터베이스를 구축할 수 있을 뿐만 아니라 통상적인 데이터베이스 운영에 관련된 작업 부하를 줄일 수 있음
    - Amazon RDS에는 데이터베이스가 설치된 OS에 로그인할 수 없는 등 데이터베이스 환경 관련 제약 사항이 존재
    - 데이터베이스 관리용 서버로 Bastion Host를 구축하여 사용
      - ◆ Bastion Host란 외부에서 내부 네트워크에 접근할 수 있는 유일한 방법인 접근점
      - ◆ 보안성이 높은 인프라와 외부 인터넷을 연결하는 중계 서버로 작동하며 모든 인바운드 트래픽은 Bastion Host를 통과해야 내부 네트워크로 들어갈 수 있음
    - Bastion Host는 AWS 가상 서버인 EC2로 구축

쿠버네티스 환경 구축 과 배포

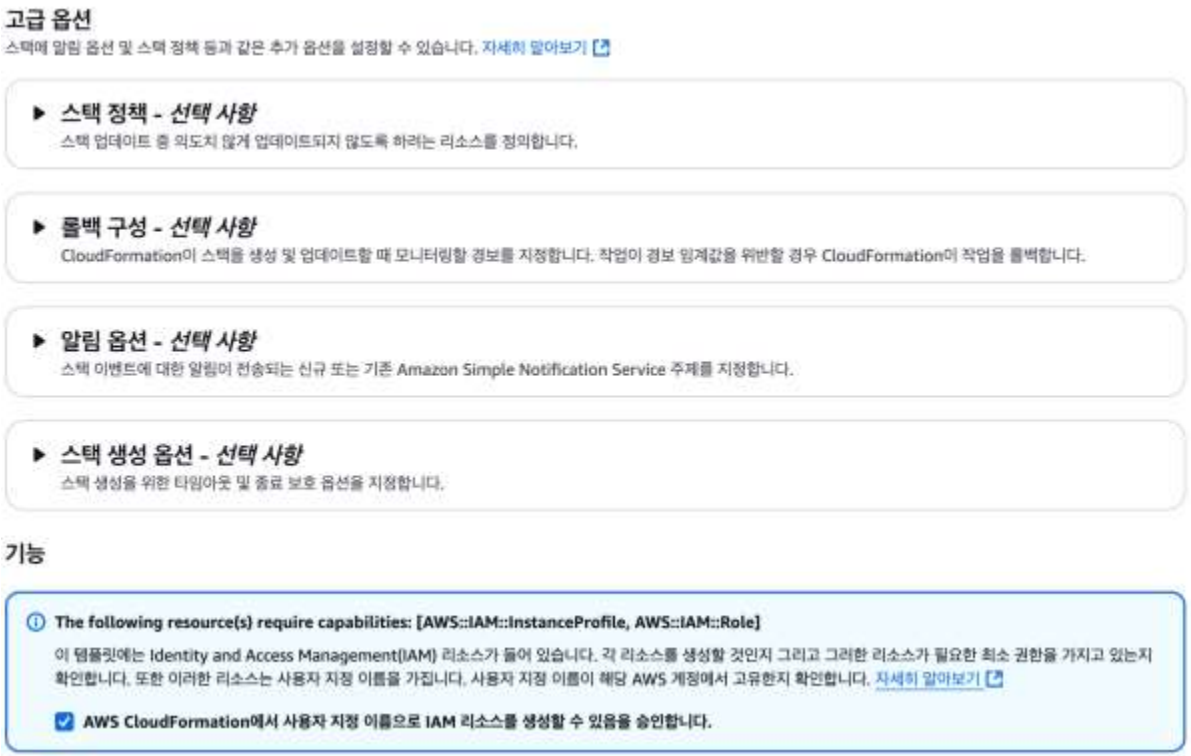
- ❖ 데이터베이스 설정
  - ✓ 데이터베이스 환경 구축
    - AWS 관리 콘솔에서 CloudFormation 페이지를 열어 [스택 생성] 버튼을 클릭하고 표시된 메뉴에서 [새 리소스 사용(표준)]을 선택하면 스택 생성 페이지가 표시
    - [준비된 템플릿] – [템플릿 파일 업로드]를 선택하고 [파일 선택] 버튼을 클릭하여 rds_ope_cfn.yaml을 선택한 후 [다음] 버튼을 클릭

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 데이터베이스 환경 구축
    - 다음 페이지에서는 스택 이름과 파라미터를 설정
    - 스택 이름에는 eks-work-rds 라고 입력
    - 업로드한 CloudFormation 템플릿에는 많은 파라미터가 정의되어 있지만 초깃값이 설정되어 있지 않아 별도의 항목을 입력해야 함
      - ◆ EksWorkVPC 풀다운 메뉴 값 중에 eks-work-vpc가 포함된 행 선택
      - ◆ OpeServeRouteTable은 CloudFormation -> eks-work-base 스택 -> 출력 탭 -> RouteTable값 입력

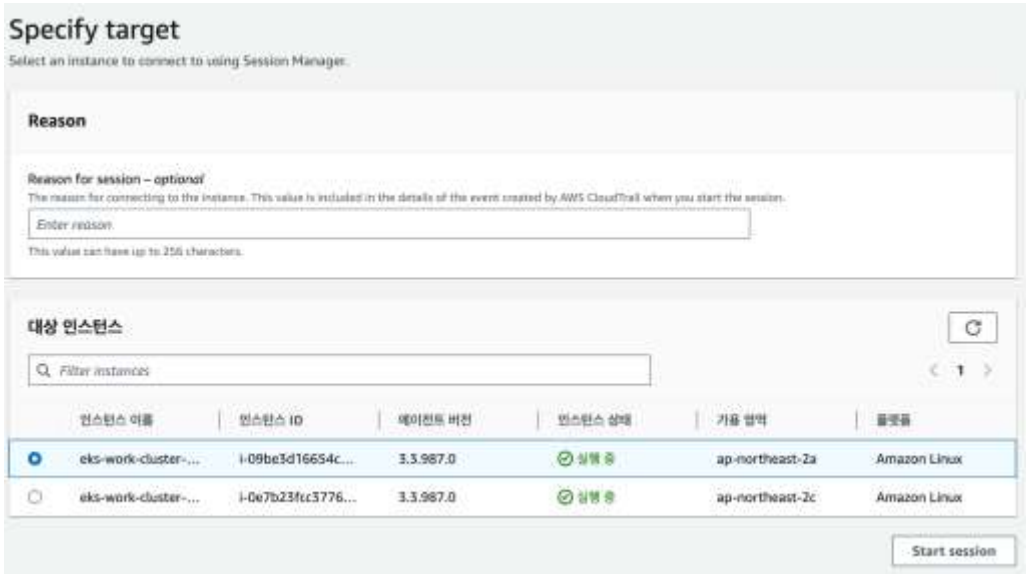
쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 데이터베이스 환경 구축
    - 나머지 옵션은 기본값을 선택하고 AWS CloudFormation에서 사용자 지정 이름으로 IAM 리소스를 생성할 수 있음을 승인합니다 부분만 체크



쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - 세션 관리자는 EC2 인스턴스 콘솔로 접속할 수 있는 서비스이며 AWS Systems Manager 기능의 하나로 제공
    - AWS 관리 콘솔에서는 [서비스] -> [관리 및 거버넌스] -> [Systems Manager]
    - 세션 시작을 눌러서 인스턴스를 선택

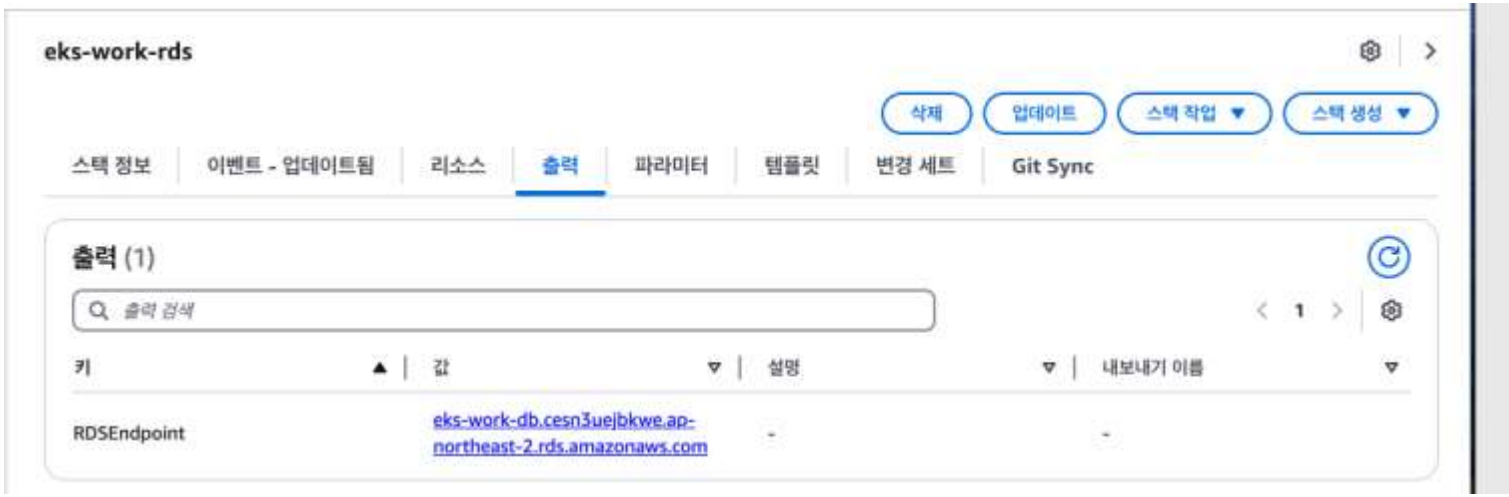


쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 도구 설치
      - ◆ Git: sudo yum install –y git
      - ◆ PostgreSQL: sudo amazon-linux-extras install -y postgresql11

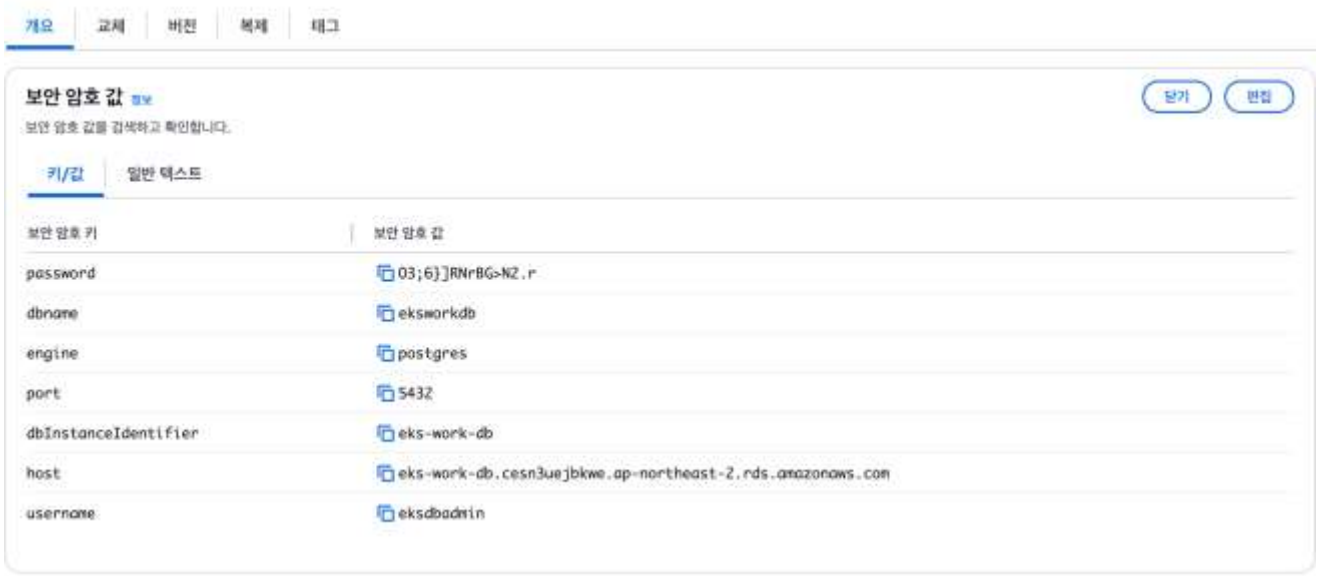
쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 엔드포인트 주소와 관리자 비밀번호 확인
      - ◆ 데이터베이스 엔드포인트: CloudFormation에서 eks-work-rds 스택을 선택한 후 출력 탭을 선택해서 RDSEndpoint 확인



쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 엔드포인트 주소와 관리자 비밀번호 확인
      - ◆ 데이터베이스 관리자 비밀번호 확인: CloudFormation으로 데이터베이스 RDS 인스턴스를 구축한 경우에는 AWS Secrets Manager가 RDS 관리자 비밀번호를 생성해 데이터베이스에 등록하는데 AWS Secrets Manager로 생성한 비밀번호는 보안 암호 에서 확인



쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 데이터베이스 사용자 생성

      createuser -d -U eksdbadmin -P -h <RDS 엔드포인트 주소> <사용자 - mywork>
        - ▪ 실행하면 비밀번호를 입력할 프롬프트가 총 세 번 표시
        - ▪ 첫 두 번의 프롬프트는 신규로 생성할 사용자 mywork의 비밀번호를 입력하는데 AWS Secrets Manager의 보안 암호로 확인한 데이터베이스 사용자의 비밀번호 사용
        - ▪ 세 번째 프롬프트는 데이터베이스 관리자인 eksdbadmin의 비밀번호를 입력

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 데이터베이스 생성

        createdb -U <사용자 – mywork> -h <RDS 엔드포인트 주소> -E UTF8 <데이터베이스 이름 – myworkdb>
        - ▪ 앞 명령을 실행하면 비밀번호를 입력하라는 프롬프트가 표시되므로 앞에서 등록한 사용자의 비밀번호를 입력

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 데이터베이스 접속

      psql -U <사용자 – mywork> -h <RDS 엔드포인트 주소> <데이터베이스 – myworkdb>

```
sh-4.2$ psql -U mywork -h eks-work-db.cesn3uejbkwe.ap-northeast-2.rds.amazonaws.com myworkdb
Password for user mywork:
psql (11.20, server 11.22)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
```

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 테이블 생성

```
CREATE TABLE region
(
  region_id           SERIAL PRIMARY KEY,
  region_name         VARCHAR(100) NOT NULL,
  creation_timestamp TIMESTAMP     NOT NULL
);
```

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 샘플 데이터 생성

```
-- REGION
INSERT INTO region (region_name, creation_timestamp)
VALUES ('서울', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('강릉', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('대전', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('광주', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('대구', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('부산', current_timestamp);
```

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 샘플 데이터 생성

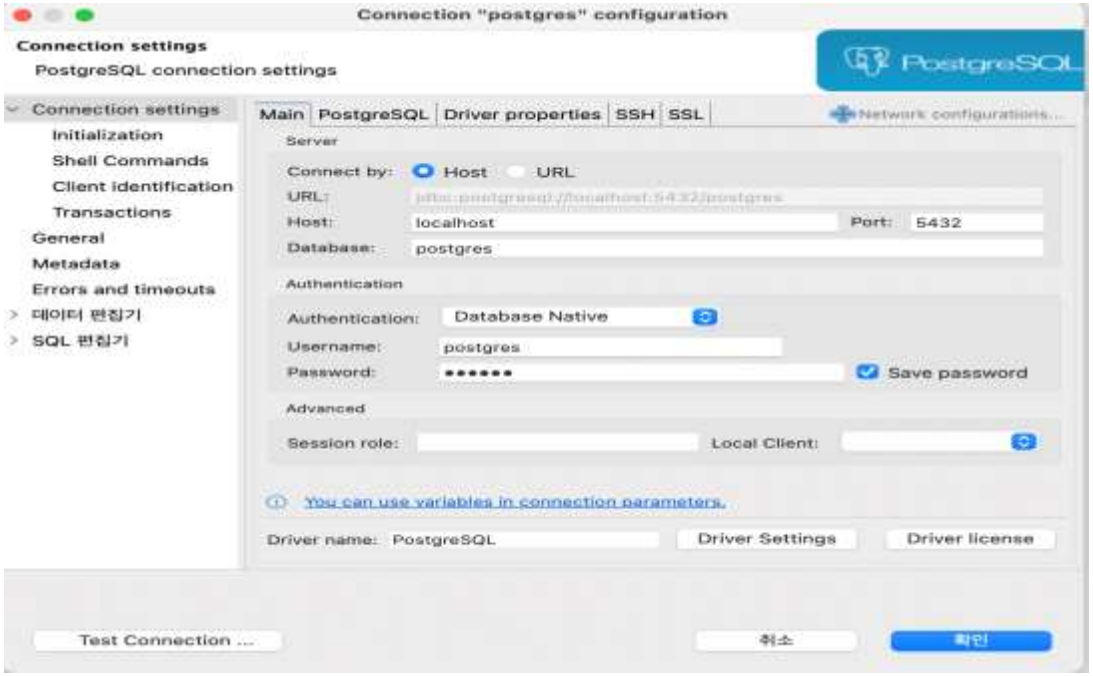
```
INSERT INTO region (region_name, creation_timestamp)
VALUES ('여수', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('안동', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('제주도', current_timestamp);
```

쿠버네티스 환경 구축 과 배포

- ❖ 데이터베이스 설정
  - ✓ 세션 관리자를 통한 배스천 호스트 접속
    - ● 데이터베이스 작업
      - ◆ 샘플 데이터 생성: sql 확장자로 스크립트 파일을 만들고 ₩i 파일경로 를 이용해서 스크립트 실행 가능
      - ◆ 접속 종료: ₩q

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 테스트를 위한 데이터베이스 작업
    - 로컬에 postgresql을 설치: docker run -d -p 외부포트번호:5432 -e POSTGRES_PASSWORD= " 비밀번호" --name 컨테이너이름 postgres
    - 접속 확인



쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 테스트를 위한 데이터베이스 작업
    - ● 사용자 및 데이터베이스 생성

```
CREATE user mywork PASSWORD 'wnddkd' SUPERUSER;

CREATE DATABASE myworkdb OWNER mywork;
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 테스트를 위한 데이터베이스 작업
    - ● 생성한 유저로 재접속을 하고 샘플 테이블 생성

```
CREATE user mywork PASSWORD 'wnddkd' SUPERUSER;

CREATE DATABASE myworkdb OWNER mywork;
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 테스트를 위한 데이터베이스 작업
    - ● 테이블 생성

```
CREATE TABLE region
(
  region_id           SERIAL PRIMARY KEY,
  region_name         VARCHAR(100) NOT NULL,
  creation_timestamp TIMESTAMP     NOT NULL
);
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 테스트를 위한 데이터베이스 작업
    - ● 샘플 데이터 생성

```
-- REGION
INSERT INTO region (region_name, creation_timestamp)
VALUES ('서울', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('강릉', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('대전', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('광주', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('대구', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('부산', current_timestamp);
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 테스트를 위한 데이터베이스 작업
    - ● 샘플 데이터 생성

```
INSERT INTO region (region_name, creation_timestamp)
VALUES ('여수', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('안동', current_timestamp);

INSERT INTO region (region_name, creation_timestamp)
VALUES ('제주도', current_timestamp);
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션
    - ● Spring Boot Application 생성
      - ◆ 의존성 추가
        - ▪ Spring Dev Tools
        - ▪ Lombok
        - ▪ Data JPA
        - ▪ Spring Web
        - ▪ Spring PostgreSQL

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션
    - ● Spring Boot Application 생성
      - ◆ 프로젝트 의존성 설정 – build.gradle

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'

    implementation 'org.apache.commons:commons-lang3:3.9'
    testImplementation('org.springframework.boot:spring-boot-starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
    }
    testImplementation 'com.ninja-squad:DbSetup:2.1.0'
    testImplementation 'org.assertj:assertj-db:1.3.0'

    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션
    - ● 프로젝트 설정 정보
      - ◆ application.properties

```
spring.application.name=backend-app

spring.datasource.url=jdbc:postgresql://localhost:5432/myworkdb
spring.datasource.username=mywork
spring.datasource.password=wnddkd
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션
    - ● 프로젝트 설정 정보
      - ◆ resources 디렉토리에 logback.xml 파일을 생성해서 작성

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="STDOUT"
class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-
5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 테이블 과 연동되는 Entity 작업
      - ◆ Entity의 공통 메서드를 가진 클래스 - AbstractEntity

```
import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;
import org.apache.commons.lang3.builder.ToStringBuilder;

public abstract class AbstractEntity {

    @Override
    public String toString() {
        return ToStringBuilder.reflectionToString(this);
    }
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 테이블 과 연동되는 Entity 작업
      - ◆ Entity의 공통 메서드를 가진 클래스 - AbstractEntity

```
    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (obj == this) {
            return true;
        }
        if (obj.getClass() != getClass()) {
            return false;
        }
        return EqualsBuilder.reflectionEquals(this, obj);
    }

    @Override
    public int hashCode() {
        return HashCodeBuilder.reflectionHashCode(this);
    }
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 테이블 과 연동되는 Entity 작업
      - ◆ RegionEntity

```
import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "REGION")
public class RegionEntity extends AbstractEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "REGION_ID")
    private Integer regionId;

    @Column(name = "REGION_NAME")
    private String regionName;

    @Column(name = "CREATION_TIMESTAMP")
    private LocalDateTime creationTimestamp;
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 테이블 과 연동되는 Entity 작업
      - ◆ RegionEntity

```
public Integer getRegionId() {
    return regionId;
}

public void setRegionId(Integer regionId) {
    this.regionId = regionId;
}

public String getRegionName() {
    return regionName;
}

public void setRegionName(String regionName) {
    this.regionName = regionName;
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 테이블 과 연동되는 Entity 작업
      - ◆ RegionEntity

```
        public LocalDateTime getCreationTimestamp() {
            return creationTimestamp;
        }

        public void setCreationTimestamp(LocalDateTime creationTimestamp) {
            this.creationTimestamp = creationTimestamp;
        }

    }
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 데이터베이스 작업을 위한 Repository 인터페이스
      - ◆ RegionRepository

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface RegionRepository extends JpaRepository<RegionEntity,
Integer> {
    Optional<RegionEntity> findByRegionName(String regionName);
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 데이터베이스 작업을 위한 Repository 테스트
      - ◆ RegionRepositoryTest

```
import com.ninja_squad.dbsetup.DbSetup;
import com.ninja_squad.dbsetup.destination.DataSourceDestination;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.test.context.SpringBootTest;

import javax.sql.DataSource;
import java.time.LocalDateTime;

import static com.ninja_squad.dbsetup.Operations.*;
import static org.assertj.core.api.Assertions.assertThat;
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 데이터베이스 작업을 위한 Repository 테스트
      - ◆ RegionRepositoryTest

```
@SpringBootTest
public class RegionRepositoryTest {

    @Autowired
    private RegionRepository repository;

    @Autowired
    @Qualifier("dataSource")
    private DataSource dataSource;

    @Test
    @Tag("DBRequired")
    public void testFindAll() {
        prepareDatabase();

        var result = repository.findAll();
        assertThat(result).hasSize(4);
    }
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 데이터베이스 작업을 위한 Repository 테스트
      - ◆ RegionRepositoryTest

```
@Test
@Tag("DBRequired")
public void testFindByRegionName() {
    prepareDatabase();

    var result = repository.findByRegionName("지역 1");
    assertThat(result.get().getRegionId()).isEqualTo(1);
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 데이터베이스 작업을 위한 Repository 테스트
      - ◆ RegionRepositoryTest

```
@BeforeEach
    public void prepareDatabase() {
        var operations = sequenceOf(
                deleteAllFrom("location"),
                deleteAllFrom("region"),
                insertInto("region")
                        .columns("region_id", "region_name", "creation_timestamp")
                        .values(1, "지역 1", LocalDateTime.now())
                        .values(2, "지역 2", LocalDateTime.now())
                        .values(3, "지역 3", LocalDateTime.now())
                        .values(4, "지역 4", LocalDateTime.now())
                        .build()
        );
        var dbSetup = new DbSetup(new DataSourceDestination(dataSource),
operations);
        dbSetup.launch();
    }

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 유저의 요청을 처리하기 위해서 사용하는 Domain 클래스
      - ◆ Region

```
import java.time.LocalDateTime;

public class Region {

    private Integer regionId;

    private String regionName;

    private LocalDateTime creationTimestamp;

    public Region(Integer regionId, String regionName, LocalDateTime
creationTimestamp) {
        if (regionName == null) {
            throw new IllegalArgumentException("regionName cannot be null.");
        }
        this.regionId = regionId;
        this.regionName = regionName;
        this.creationTimestamp = creationTimestamp;
    }
```

쿠버네티스 환경 구축 과 배포

- 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 유저의 요청을 처리하기 위해서 사용하는 Domain 클래스
      - ◆ Region

```
public Region(RegionEntity entity) {
    this(entity.getRegionId(), entity.getRegionName(),
entity.getCreationTimestamp());
}

public Integer getRegionId() {
    return regionId;
}

public void setRegionId(Integer regionId) {
    this.regionId = regionId;
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 유저의 요청을 처리하기 위해서 사용하는 Domain 클래스
      - ◆ Region

```
    public String getRegionName() {
        return regionName;
    }

    public void setRegionName(String regionName) {
        this.regionName = regionName;
    }

    public LocalDateTime getCreationTimestamp() {
        return creationTimestamp;
    }

    public void setCreationTimestamp(LocalDateTime creationTimestamp) {
        this.creationTimestamp = creationTimestamp;
    }
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● 유저의 요청을 처리하기 위한 Service 클래스
      - ◆ RegionService

```
import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
public class RegionService {

    private final RegionRepository regionRepository;

    public List<Region> getAllRegions() {
        var regionEntities = regionRepository.findAll();
        var regionList = new ArrayList<Region>();
        regionEntities.forEach(entity -> regionList.add(new Region(entity)));

        return regionList;
    }

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Service 클래스
      - ◆ RegionService

```
import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
public class RegionService {

    private final RegionRepository regionRepository;

    public List<Region> getAllRegions() {
        var regionEntities = regionRepository.findAll();
        var regionList = new ArrayList<Region>();
        regionEntities.forEach(entity -> regionList.add(new Region(entity)));

        return regionList;
    }

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 에서 데이터 리턴을 하기 위한 클래스
      - ◆ HealthDto

```
public class HealthDto {

    private String status;

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 에서 데이터 리턴을 하기 위한 클래스
      - ◆ RegionDto

```
public class RegionDto {

    private Integer regionId;

    private String regionName;

    public RegionDto() {
    }

    public RegionDto(Region region) {
        this.regionId = region.getRegionId();
        this.regionName = region.getRegionName();
    }
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 에서 데이터 리턴을 하기 위한 클래스
      - ◆ RegionDto

```
        public Integer getRegionId() {
            return regionId;
        }

        public void setRegionId(Integer regionId) {
            this.regionId = regionId;
        }

        public String getRegionName() {
            return regionName;
        }

        public void setRegionName(String regionName) {
            this.regionName = regionName;
        }
    }
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 에서 데이터 리턴을 하기 위한 클래스
      - ◆ RegionsDto

```
import java.util.ArrayList;
import java.util.List;
public class RegionsDto {

    private List<RegionDto> regionList = new ArrayList<>();

    public RegionsDto() {
    }

    public RegionsDto(List<RegionDto> regionDtoList) {
        this.regionList.addAll(regionDtoList);
    }
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 에서 데이터 리턴을 하기 위한 클래스
      - ◆ RegionsDto

```
    public List<RegionDto> getRegionList() {
        return regionList;
    }

    public void setRegionList(List<RegionDto> regionList) {
        this.regionList = regionList;
    }
}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 클래스
      - ◆ HealthApi

```
@RestController
@RequestMapping("health")
public class HealthApi {

    private static final Logger LOGGER =
LoggerFactory.getLogger(HealthApi.class);

    @GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public HealthDto getHealth() {
        LOGGER.info("Health GET API called.");

        var health = new HealthDto();
        health.setStatus("OK");
        return health;
    }

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 클래스
      - ◆ RegionApi

```
@RestController
@RequestMapping("region")
@CrossOrigin(origins = "*")
@RequiredArgsConstructor
public class RegionApi {

    private static final Logger LOGGER =
LoggerFactory.getLogger(RegionApi.class);

    private final RegionService service;
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 클래스
      - ◆ RegionApi

```
@GetMapping(produces = MediaType.APPLICATION_JSON_VALUE)
public RegionsDto getAllRegions() {
    LOGGER.info("REGION GET ALL API");

    var allRegions = service.getAllRegions();
    var dtoList = new ArrayList<RegionDto>();
    allRegions.forEach(region -> {
        var dto = new RegionDto(region);
        dtoList.add(dto);
    });
    var regionsDto = new RegionsDto(dtoList);
    return regionsDto;
}

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 클래스 테스트
      - ◆ HealthApiTest

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import static org.assertj.core.api.AssertionsForClassTypes.assertThat;

@SpringBootTest
public class HealthApiTest {

    @Test
    public void testHealthOk() {
        var api = new HealthApi();
        var health = api.getHealth();
        assertThat(health.getStatus()).isEqualTo("OK");
    }

}
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 생성
    - ● Controller 클래스 테스트
      - ◆ RegionApiTest

```
@SpringBootTest
public class RegionApiTest {
    @Autowired
    private RegionApi api;

    @Mock
    private RegionRepository repository;


    @Test
    public void testGetAllRegions() {
        var result = api.getAllRegions();
        assertThat(result.getRegionList()).hasSize(4);
    }
}
```

쿠버네티스 환경 구축 과 배포

- 애플리케이션 생성
  - ✓ 배포를 위해서 외부 데이터베이스를 사용할 수 있도록 application.properties 수정

```
spring.application.name=backend-app

spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

쿠버네티스 환경 구축 과 배포

- 애플리케이션 생성
  - ✓ 데이터베이스 접속 정보를 숨기기 위해서 application.properties 수정

```
spring.application.name=backend-app

spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 애플리케이션 빌드
    - ● 빌드 시 수행되는 작업
      - ◆ 의존성 라이브러리 다운로드
      - ◆ 프로그램 컴파일
      - ◆ 테스트 프로그램 컴파일
      - ◆ 테스트 실행
      - ◆ 프로그램 실행용 아카이브 파일 (JAR 파일) 생성
    - ● 명령어
      - ◆ sudo chmod 755 ./gradlew
      - ◆ ./gradlew clean build

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 도커 이미지를 ECR에 업로드
    - Dockerfile 생성

```
FROM amazoncorretto:17

CMD ["./mvnw", "clean", "package"]

ARG JAR_FILE=target/*.jar

COPY ./build/libs/*.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 도커 이미지를 ECR에 업로드
    - ● 이미지 빌드

```
sudo docker build -t k8s/backend-app:1.0.0 --build-arg
JAR_FILE=build/libs/backend-app-0.0.1-SNAPSHOT.jar .
```

쿠버네티스 환경 구축 과 배포

- ❖ 애플리케이션 생성
  - ✓ 도커 이미지를 ECR에 업로드
    - ECR에 레포지토리 생성: k8s/backend-app
    - ECR에 로그인: aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
    - 이미지 태그 변경: docker tag k8s/backend-app:1.0.0 641022061021.dkr.ecr.ap-northeast-2.amazonaws.com/k8s/backend-app:1.0.0
    - 도커 푸시: docker push 641022061021.dkr.ecr.ap-northeast-2.amazonaws.com/k8s/backend-app:1.0.0

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 작업
    - 네임스페이스 생성
    - kubeconfig에 네임스페이스 반영
    - 데이터베이스 접속용 시크릿 등록
    - API 애플리케이션 배포
    - API 애플리케이션 외부 공개

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 네임스페이스 생성
    - ● 네임스페이스
      - ◆ 쿠버네티스에서는 클러스터 하나를 네임스페이스라는 논리적 구획으로 구분하여 관리할 수 있음
      - ◆ eks-work라는 네임스페이스를 생성하고 그 네임 스페이스에 API 애플리케이션을 배포
      - ◆ 네임스페이스는 쿠버네티스 오브젝트 설정 파일을 kubectl apply 명령으로 적용하여 생성
      - ◆ 쿠버네티스 오브젝트는 kubectl create 명령으로도 생성 가능하지만 보통 매니페스트 파일을 이용해서 생성하는데 원하는 인프라 환경 상태를 소스 코드와 비슷한 매니페스트 파일로 만든 후 깃 등의 버전 관리 도구로 관리할 수 있기 때문인데 인프라 환경을 한 번 구축해두면 그 설정 파일을 사용해 몇 번이고 같은 환경을 구축할 수 있음

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 네임스페이스 생성
    - ● 네임스페이스 생성
      - ◆ 야믈 파일을 생성하고 작성

```
apiVersion: v1
kind: Namespace
metadata:
  name: eks-work
```

      - ◆ 야믈 파일 실행: kubectl apply -f create_namespace_k8s.yaml

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ kubeconfig에 네임스페이스 반영
    - kubectl config get-contexts 명령을 이용해서 현재 사용 중인 컨텍스트 정보 확인

```
CURRENT   NAME                                                  CLUSTER                                       AUTHINFO
                NAMESPACE
          docker-desktop                                        docker-desktop                                docker-desktop

*         iam-root-account@eks-work-cluster.ap-northeast-2.eksctl.io   eks-work-cluster.ap-northeast-2.eksctl.io   iam-root-account@eks-work-cluster.ap-nort
heast-2.eksctl.io
          kind-kindcluster                                      kind-kindcluster                              kind-kindcluster

          minikube                                              minikube                                      minikube
                default
          rancher-desktop                                       rancher-desktop                               rancher-desktop
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ kubeconfig에 네임스페이스 반영
    - ● 현재 컨텍스트에 네임스페이스 반영
      - ◆ kubectl config set-context eks-work --cluster <클러스터 이름> --user <AUTHINFO 값> --namespace <네임스페이스 이름>

        Context "eks-work" created.
      - ◆ kubectl config use-context eks-work

        Switched to context "eks-work".
      - ◆ kubectl config get-contexts 명령을 이용해서 현재 사용 중인 컨텍스트 정보 확인

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 데이터베이스 접속용 시크릿 등록
    - 시크릿은 쿠버네티스 클러스터 안에 비밀번호 등의 비밀 정보를 보관하기 위한 구성 요소
    - 시크릿 저장을 위한 template 파일 생성: db_config_k8s.yaml.template

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: db-config
stringData:
  db-username: mywork
  db-password: ${DB_PASSWORD}
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 데이터베이스 접속용 시크릿 등록
    - ● 시크릿 저장 명령 수행: End Point 주소 와 Password는 수정

```
DB_PASSWORD='<xGN)~wb.Vg-!qgt4>' envsubst < db_config_k8s.yaml.template |
kubectl apply -f -
```

쿠버네티스 환경 구축 과 배포

❖ EKS 클러스터에 API 애플리케이션 배포

✓ 애플리케이션 배포

● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-app
  labels:
    app: backend-app
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 배포
    - ● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend-app
```

쿠버네티스 환경 구축 과 배포

❖ EKS 클러스터에 API 애플리케이션 배포

✓ 애플리케이션 배포

● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
template:
  metadata:
    labels:
      app: backend-app
  spec:
    containers:
    - name: backend-app
      image: ${ECR_HOST}/k8s/backend-app:1.0.0
      imagePullPolicy: Always
      ports:
      - containerPort: 8080
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 배포
    - ● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
env:
- name: DB_URL
  valueFrom:
    secretKeyRef:
      key: db-url
      name: db-config
- name: DB_USERNAME
  valueFrom:
    secretKeyRef:
      key: db-username
      name: db-config
```

쿠버네티스 환경 구축 과 배포

❖ EKS 클러스터에 API 애플리케이션 배포

✓ 애플리케이션 배포

● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
- name: DB_PASSWORD
  valueFrom:
    secretKeyRef:
      key: db-password
      name: db-config
readinessProbe:
  httpGet:
    port: 8080
    path: /health
  initialDelaySeconds: 15
  periodSeconds: 30
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 배포
    - ● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
livenessProbe:
  httpGet:
    port: 8080
    path: /health
  initialDelaySeconds: 30
  periodSeconds: 30
resources:
  requests:
    cpu: 100m
    memory: 512Mi
  limits:
    cpu: 250m
    memory: 768Mi
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 배포
    - ● 배포를 위한 템플릿 파일 생성 – deployment_backend_app_k8s.yaml.template

```
lifecycle:
  preStop:
    exec:
      command: ["/bin/sh", "-c", "sleep 2"]
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 배포
    - 배포 명령 수행

      ECR_HOST=<AWS 계정>.dkr.ecr.ap-northeast-2.amazonaws.com envsubst <deployment_backend-app_k8s.yaml.template | kubectl apply -f –
    - 배포 결과 확인

      kubectl get all

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 외부 공개
    - ● 개요
      - ◆ API 애플리케이션 배포가 끝났지만 아직 클러스터 외부에서는 API 호출이 불가능
      - ◆ 쿠버네티스에는 배포된 파드를 외부에 공개하기 위해 서비스 라는 리소스 이용
      - ◆ 서비스는 공개 범위에 따라 몇 가지 타입이 정의되어 있는데 외부에서 접속이 가능하도록 할려면 로드밸런서라는 서비스 타입을 이용
      - ◆ 배포한 파드 앞단에 로드밸런서를 위치시키고 인터넷에서 요청을 받으면 파드에서 동작 중인 애플리케이션을 호출

쿠버네티스 환경 구축 과 배포

❖ EKS 클러스터에 API 애플리케이션 배포

✓ 애플리케이션 외부 공개

● 서비스를 위한 매니페스트 파일 생성: service_backend-app_k8s.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: backend-app-service
spec:
  type: LoadBalancer
  selector:
    app: backend-app
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
```

쿠버네티스 환경 구축 과 배포

- ❖ EKS 클러스터에 API 애플리케이션 배포
  - ✓ 애플리케이션 외부 공개
    - 서비스 실행

      kubectl apply -f service_backend-app_k8s.yaml
    - 배포 결과 확인

      kubectl get all
    - 서비스의 External IP로 외부에서 접속