

# Kubernetes Overview

Jaeyoung Ha  
Solutions Architect

# 컨테이너 오케스트레이션 필요성

# 컨테이너 오케스트레이션 툴의 필요성



- 컨테이너를 호스트에 어떻게 배포하지?
- 컨테이너 내부/외부 통신은 어떻게 하지?
- 시크릿 관리는 어떻게 하지?
- 컴퓨팅 자원 풀을 최적화 할 수 있는 방법은 무엇일까?
- 컨테이너의 생명 주기를 어떻게 관리하지?
- 무중단, 블루/그린 배포는 어떻게 할 수 있을까?

# AWS 환경 위에서의 컨테이너

## Management

컨테이너화된 애플리케이션을 배포, 스케줄링, 스케일링 및 관리



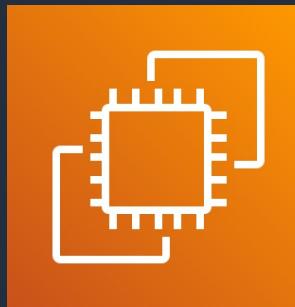
Amazon Elastic  
Container Service



Amazon Elastic  
Kubernetes Service

## Hosting

컨테이너가 실행되는 곳



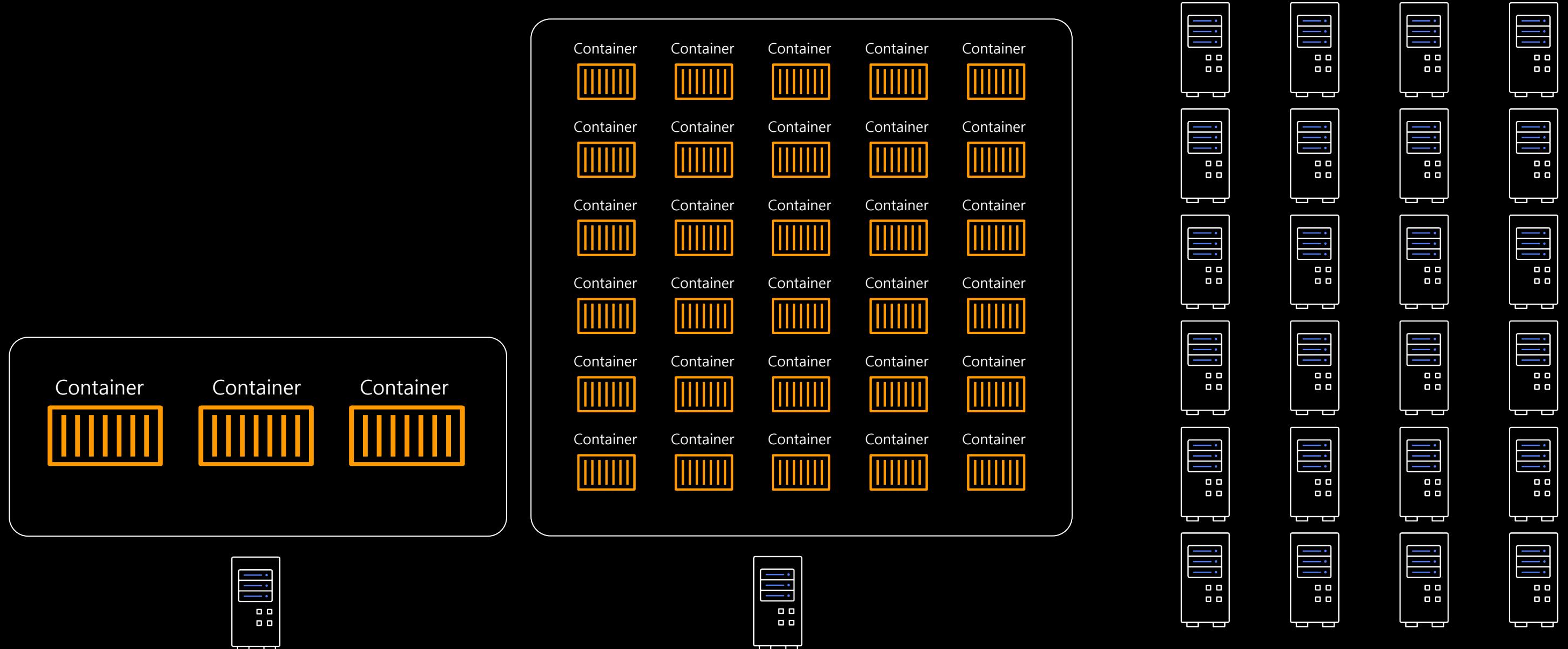
Amazon EC2



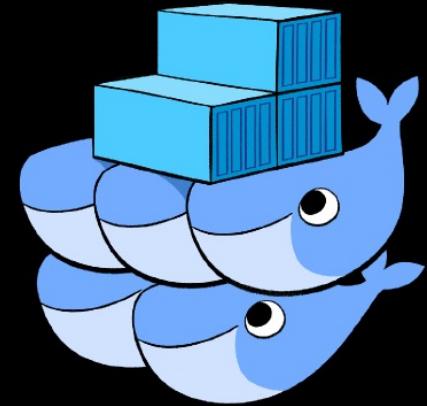
AWS Fargate

# K8S 가 필요한 이유

# Container Orchestrator 가 필요한 이유?



# Container Orchestration Tool



Docker Swarm



Kubernetes



MESOS

# Kubernetes 가 필요한 이유?

Scheduling

수 많은 서버 중 최적의 서버에 컨테이너 배포

Self Healing

배포된 컨테이너에 문제가 생겼을 경우 자동으로 대응

Load Balancing

서비스 트래픽을 배포된 컨테이너에 균등하게 분산

Auto Scaling

서비스 제공을 위해 필요한 경우 자동으로 용량 증설

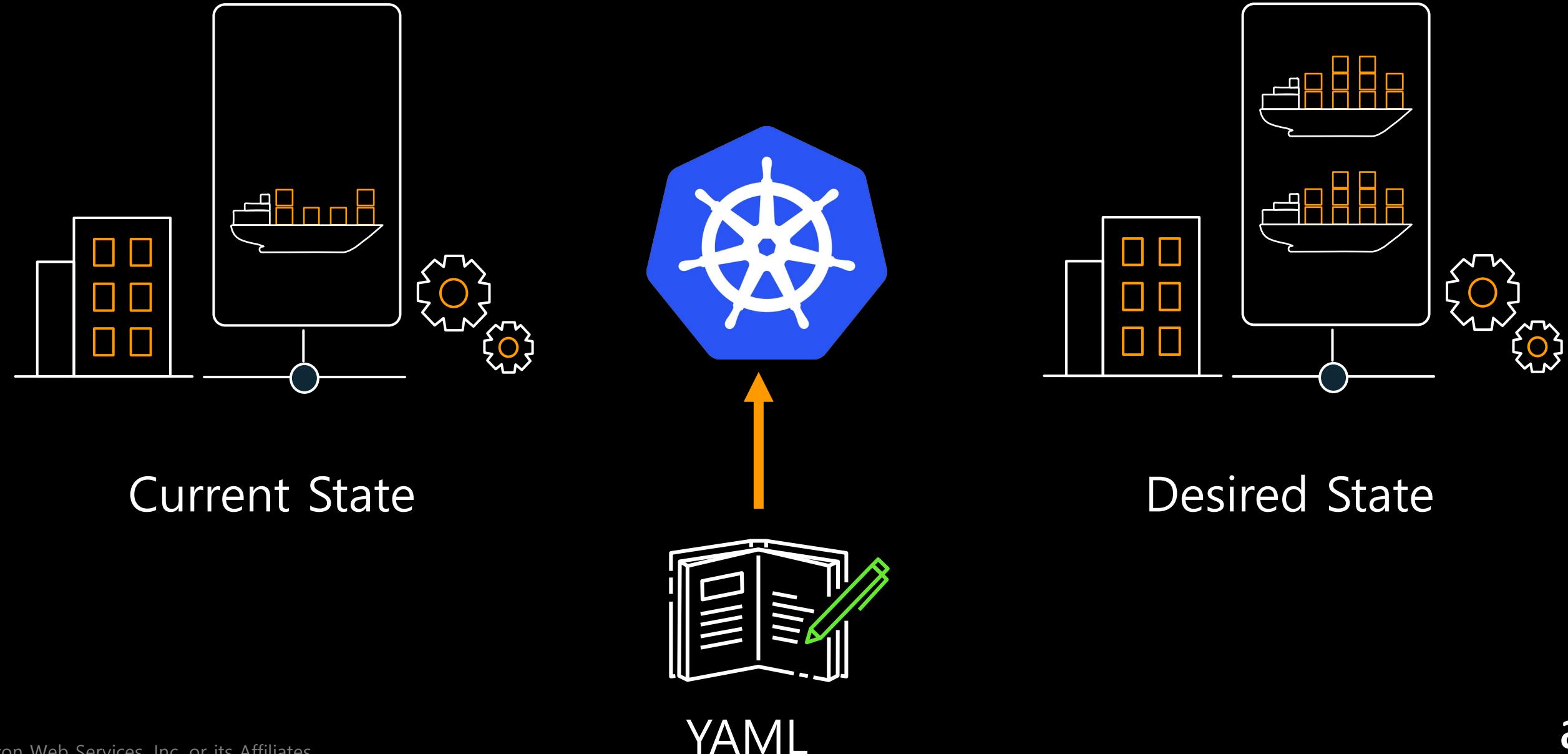
Rolling Update

배포된 컨테이너 환경에 대한 업데이트

Portability

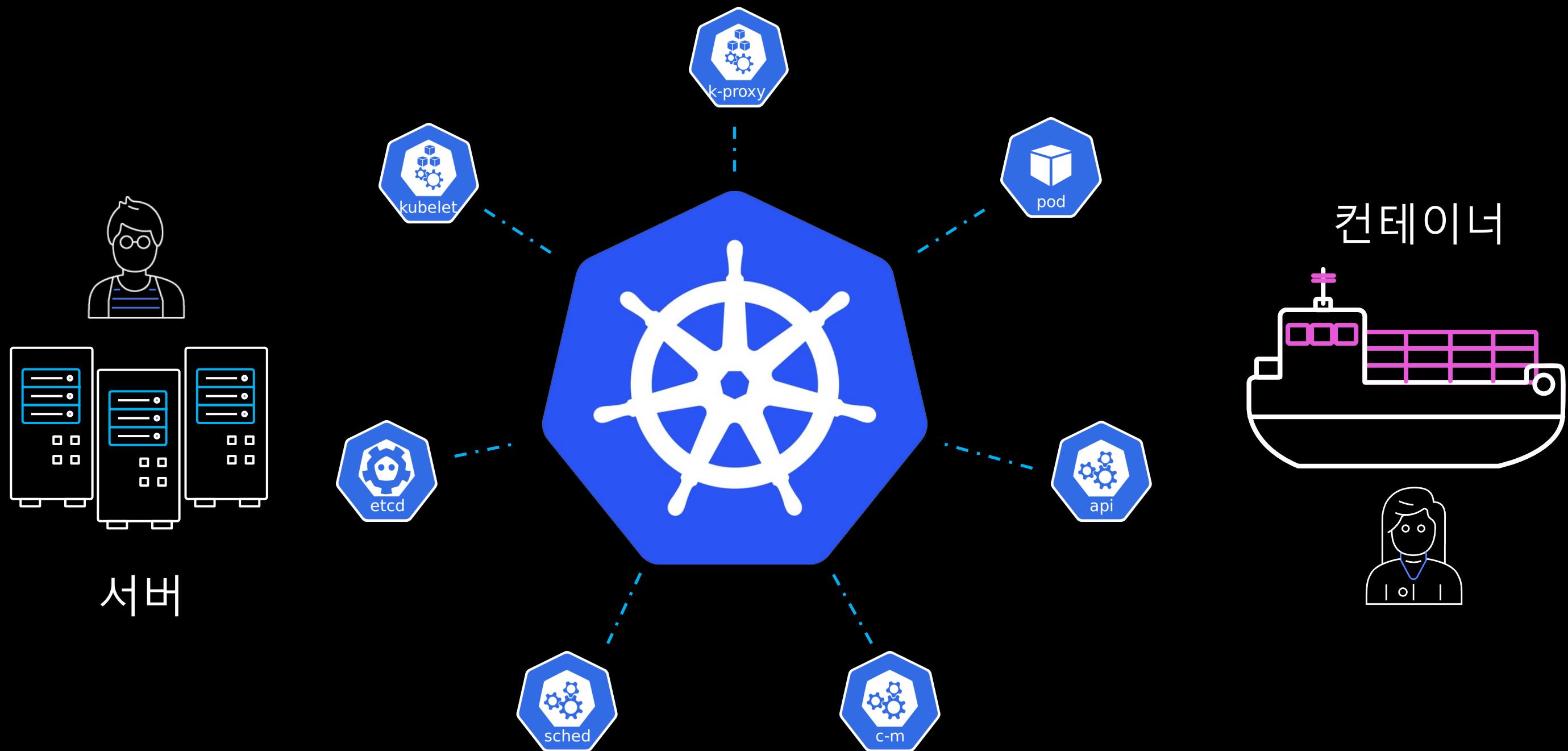
서로 다른 운영 환경에 대한 자유로운 이동

# Kubernetes 의 목표

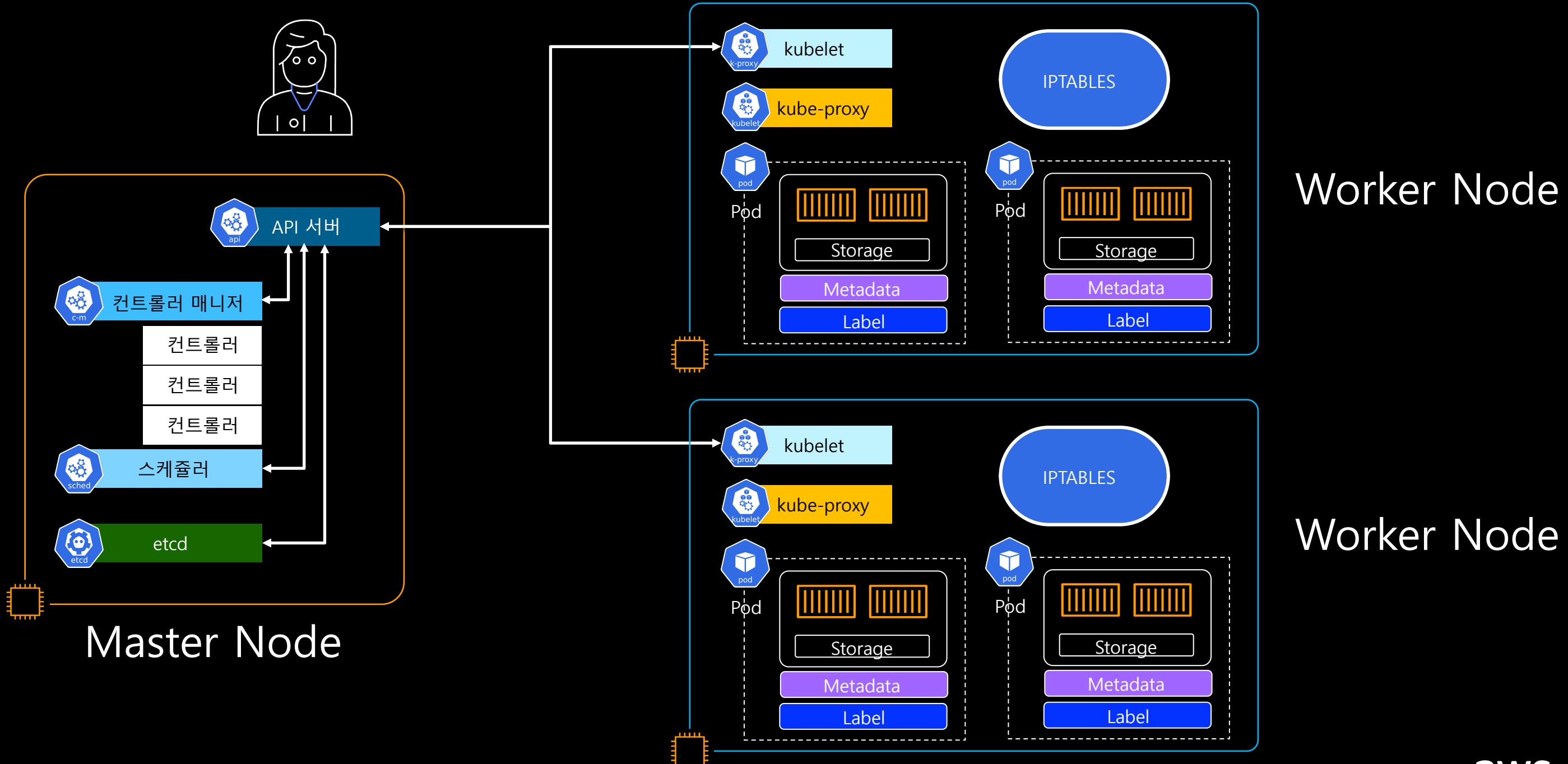


# K8S 아키텍처

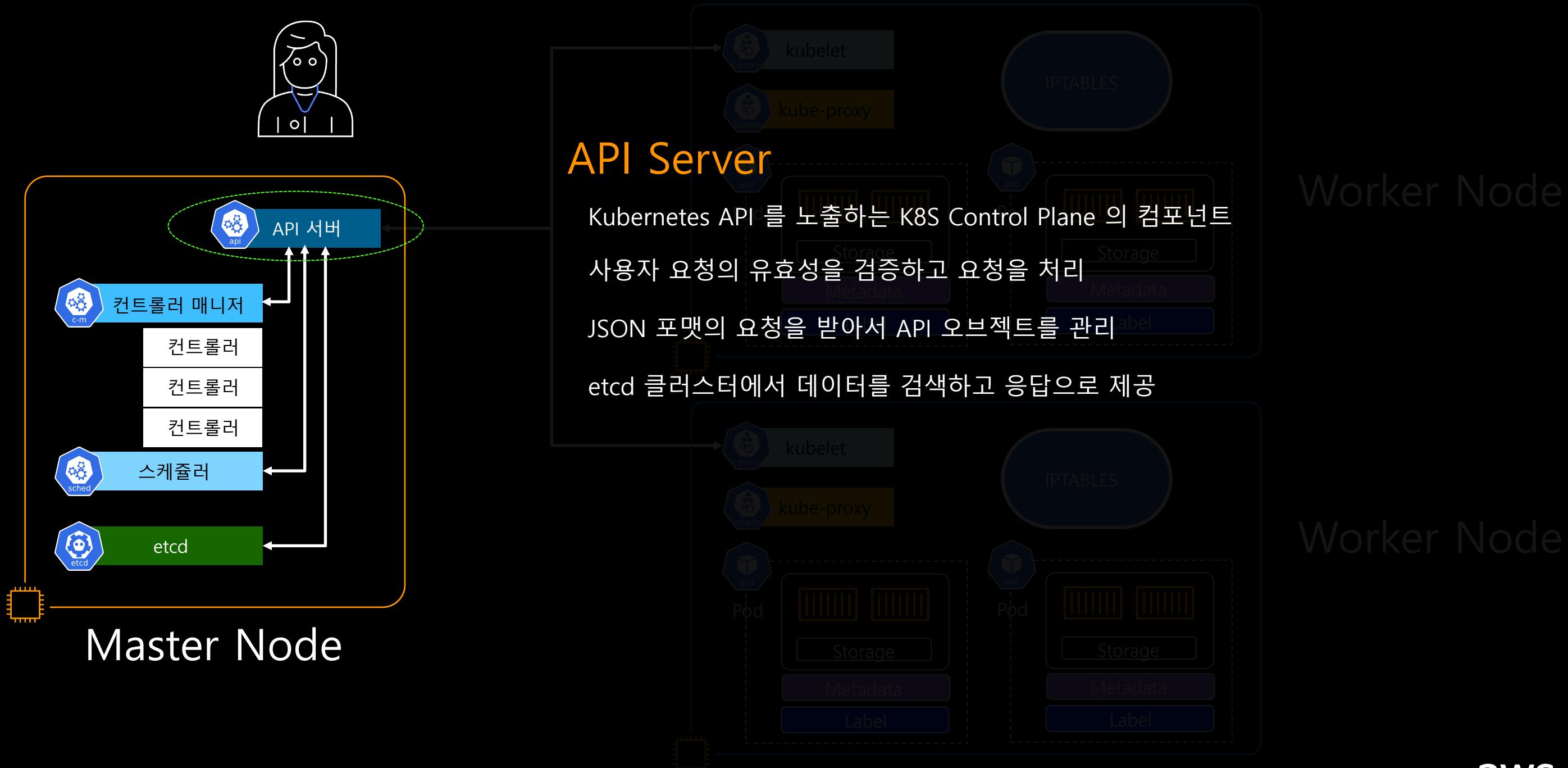
# Kubernetes, K8S, 쿠버네티스



# Kubernetes 클러스터



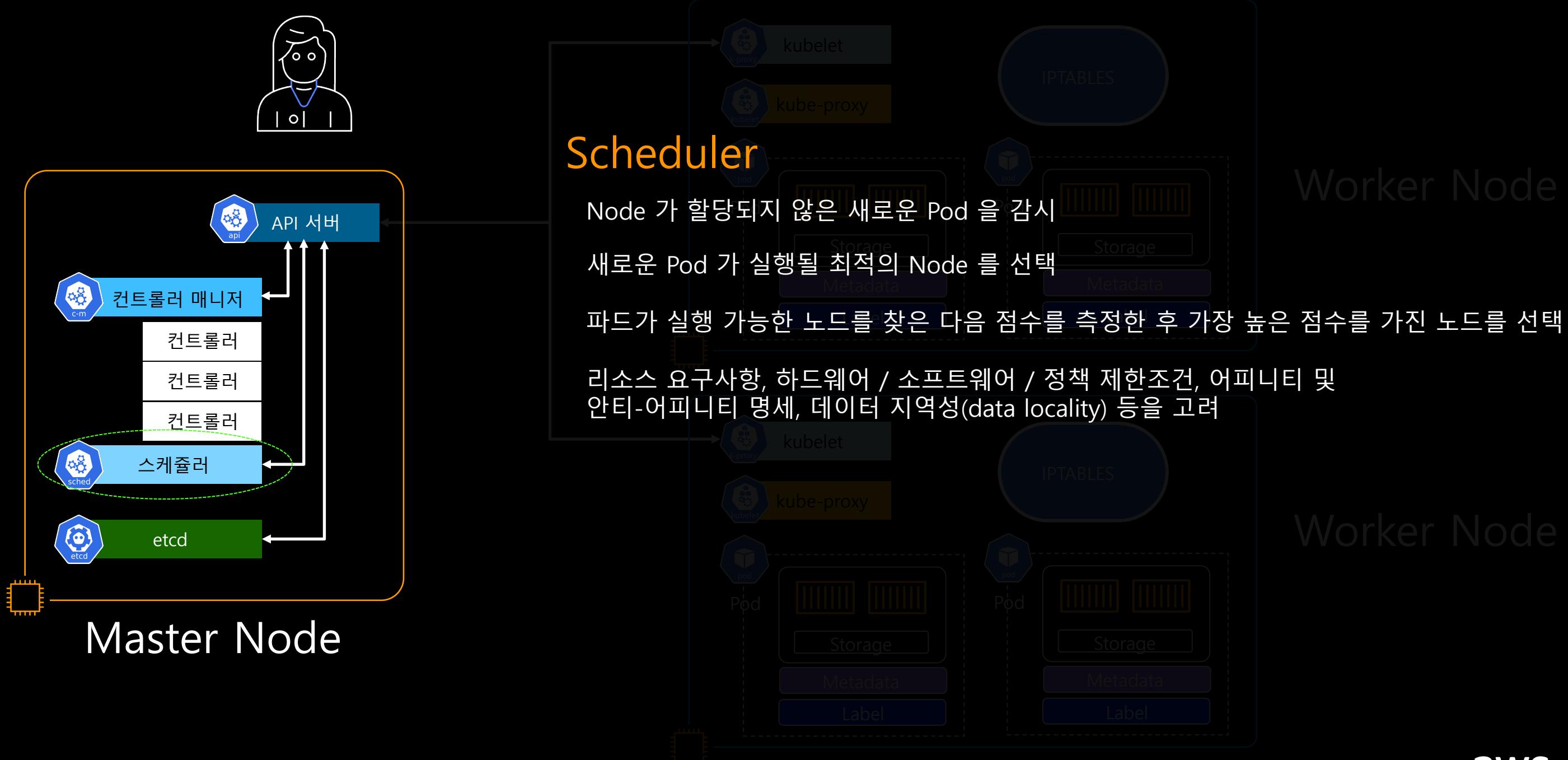
# Kubernetes 클러스터 구성요소 – Control Plane



# Kubernetes 클러스터 구성요소 – Control Plane



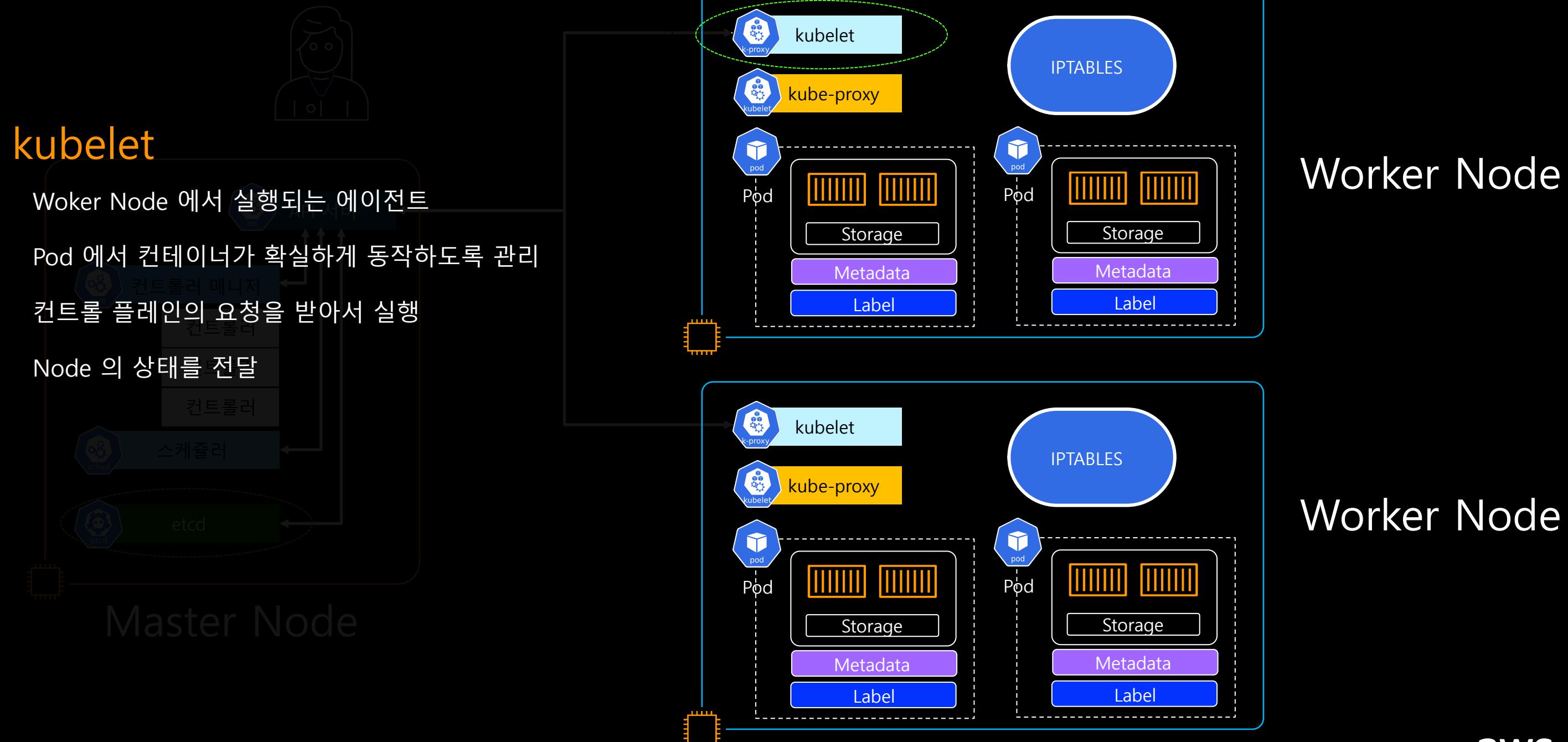
# Kubernetes 클러스터 구성요소 – Control Plane



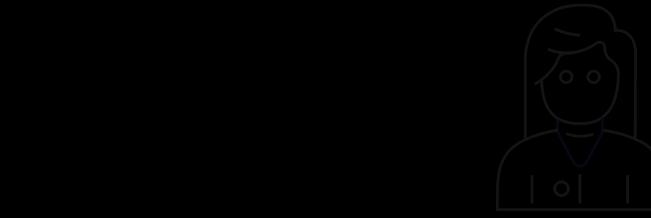
# Kubernetes 클러스터 구성요소 – Control Plane



# Kubernetes 클러스터 구성요소 - Data Plane



# Kubernetes 클러스터 구성요소 - Data Plane



## kube-proxy

Worker Node에서 실행되는 네트워크 프록시

Node의 네트워크 규칙을 유지 관리

서비스와 Endpoint에 대한 연결을 구성  
컨트롤러 매니저  
컨트롤러

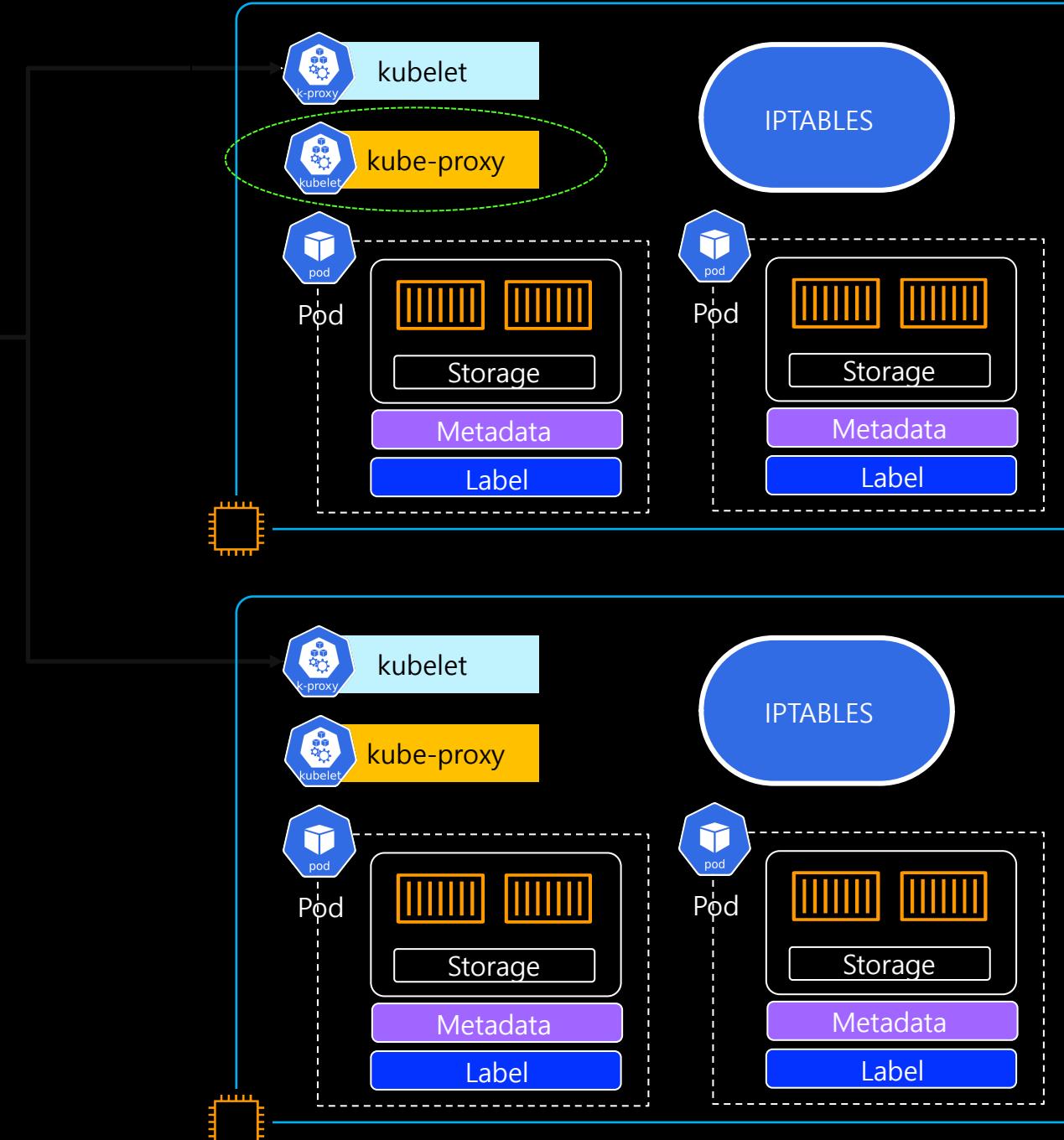
OS에서 제공하는 패킷 필터링 기능을 사용

DaemonSet 형태로 배포

스케줄러

etcd

## Master Node



## Worker Node

## Worker Node

# Kubernetes 클러스터 구성요소 - Data Plane

## Pod

Kubernetes 의 가장 작은 Object 단위

하나 이상의 컨테이너를 포함

고유의 사설 IP 를 할당 받아서 사용

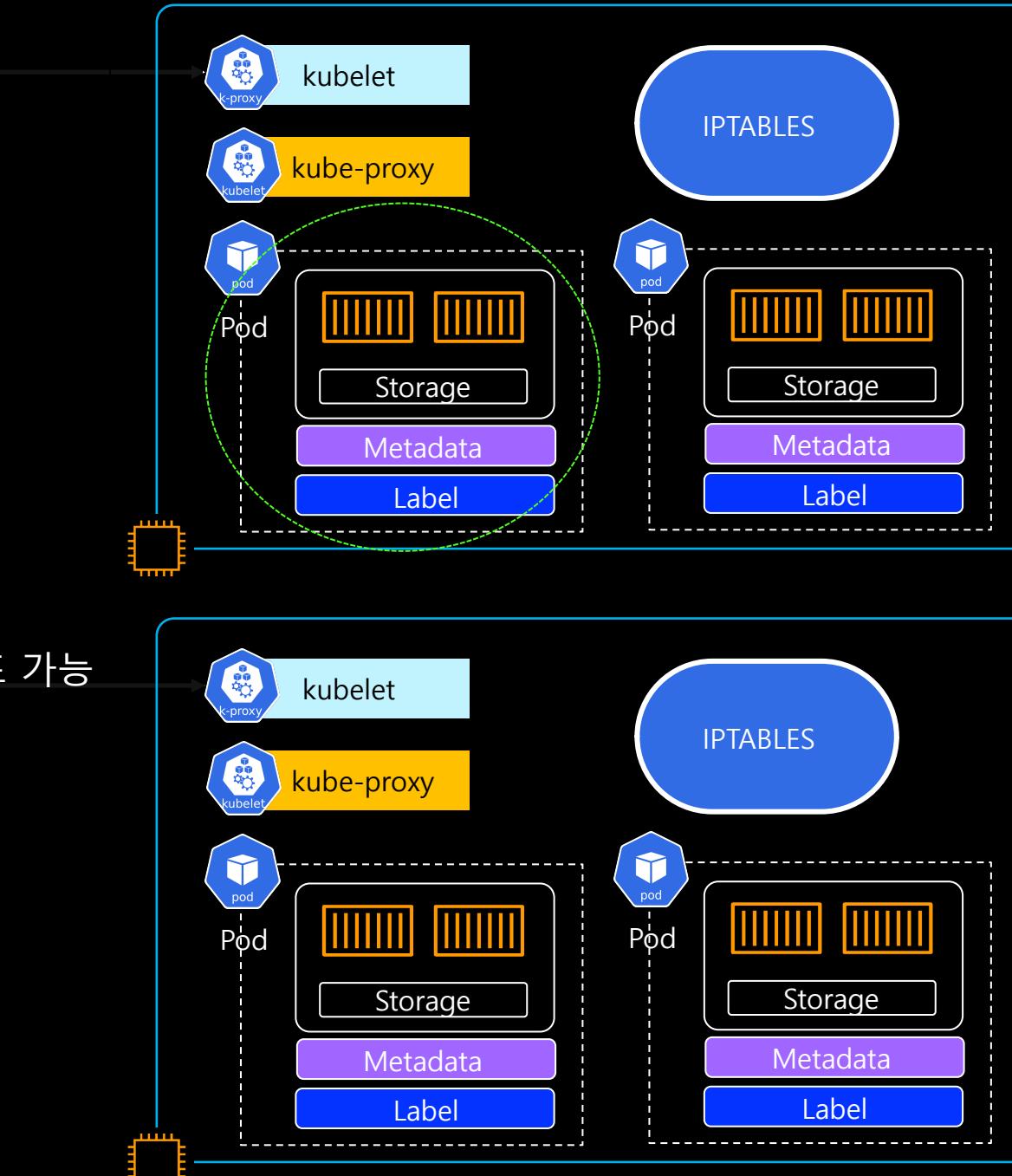
Pod 내의 컨테이너는 Pod 의 IP 를 공유해서 사용

필수 컨테이너는 Side Car 형태로 각 Pod 에 동시 배포 가능

스케줄러

etcd

## Master Node



## Worker Node

## Worker Node

# K8S Configuration

# Kubernetes Configuration

## Imperative(명령형)

무언가를 작업하기 위한 방법을 정의하는 것

run, create, expose, edit, scale, set, create -f, replace -f, delete -f

NGINX Image 를 사용하는 Pod 3개를 실행해주세요.

```
kubectl run nginxapp --image nginx:latest --replicas 3  
kubectl create service nodeport
```

정의

사용방법

예시

## Declarative(선언형)

무언가를 작업하기 위하여 어떻게 진행할 것인지를 나열하는 것

"Desired State" 를 YAML 로 정의하고 실행

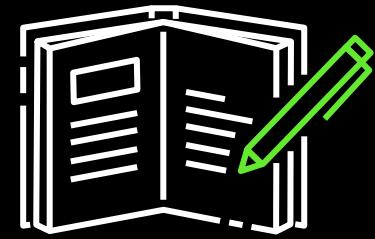
NGINX Image 를 사용하는 Pod 3개가 실행되어 있는 상태를 만들어주세요

```
kubectl apply -f nginx.yaml
```

차이점

idempotent(멱등성)

# Kubernetes YAML



Desired State

spec:

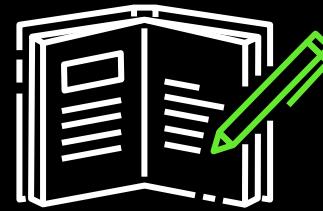
?

Current State

status:



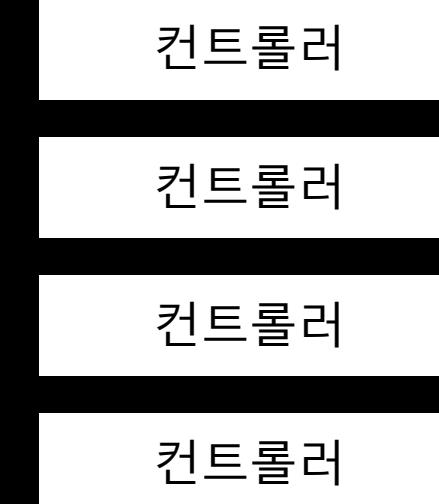
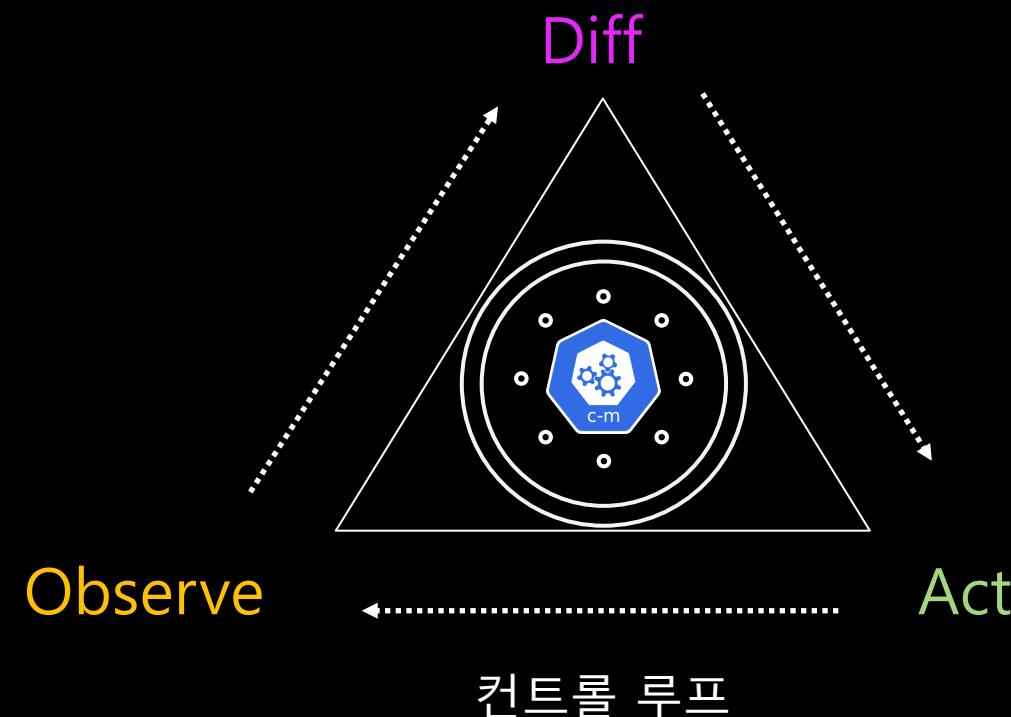
# Kubernetes Control Loop



Desired State  
spec:

?

Current State  
status:



# Kubernetes YAML

```

    ⚡ apiVersion: apps/v1
    ⚡ kind: Deployment
    ⚡ metadata:
        name: nginx-deployment
        annotations:
            description: frontend
    ⚡ labels:
        app: nginx
    spec:
        replicas: 3
        selector:
            matchLabels:
                app: nginx
        template:
            metadata:
                labels:
                    app: nginx
            spec:
                containers:
                    - name: nginx
                      image: nginx:1.7.9
                      ports:
                        - containerPort: 80
    status:

```

**apiVersion**

kind에 지정된 Kubernetes Object에 사용되는 API 버전을 지정

**kind**

YAML을 이용해 생성하고자 하는 Kubernetes Object를 지정

**metadata**

Object를 구분지어 줄 수 있는 데이터 정보 (이름, Label, Namespace 등)

**annotations**

쿠버네티스 오브젝트에 메타데이터를 첨부할 때 사용. 라벨처럼 map 형태로 구성되지만 사람보다는 machine을 위한 용도로 사용되며 검색이 불가능한 metadata를 지정하는데 사용

# Kubernetes YAML

```

    ⚡ apiVersion: apps/v1
    ⚡ kind: Deployment
    ⚡ metadata:
        name: nginx-deployment
        annotations:
            description: frontend
    ⚡ labels:
        app: nginx
    spec:
        replicas: 3
        selector:
            matchLabels:
                app: nginx
        template:
            metadata:
                labels:
                    app: nginx
            spec:
                containers:
                    - name: nginx
                      image: nginx:1.7.9
                      ports:
                        - containerPort: 80
    status:

```

**label**

Kubernetes Object 에 첨부된 key-value 값으로 오브젝트의 특성을 식별하는데 사용

**spec**

Kubernetes 에 배포될 자원에 대한 요구 사항을 명시

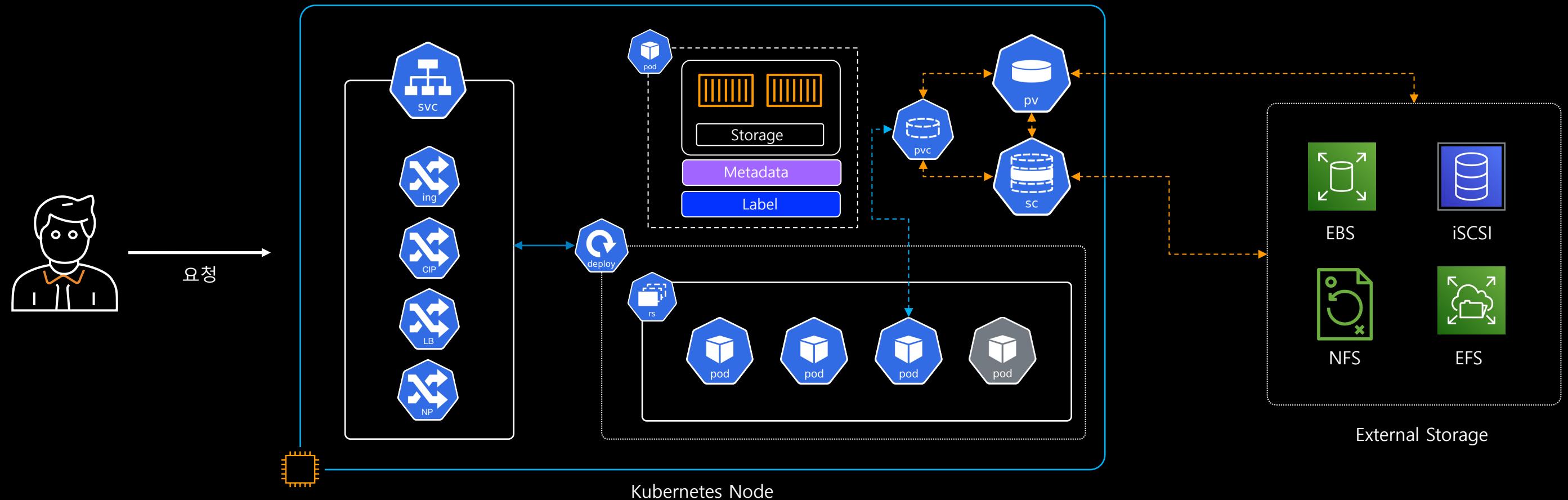
**selector**

효율적으로 레이블을 이용해 오브젝트를 쉽게 식별할 수 있게 함. 컨트롤러가 모니터링해야하는 대상을 명시

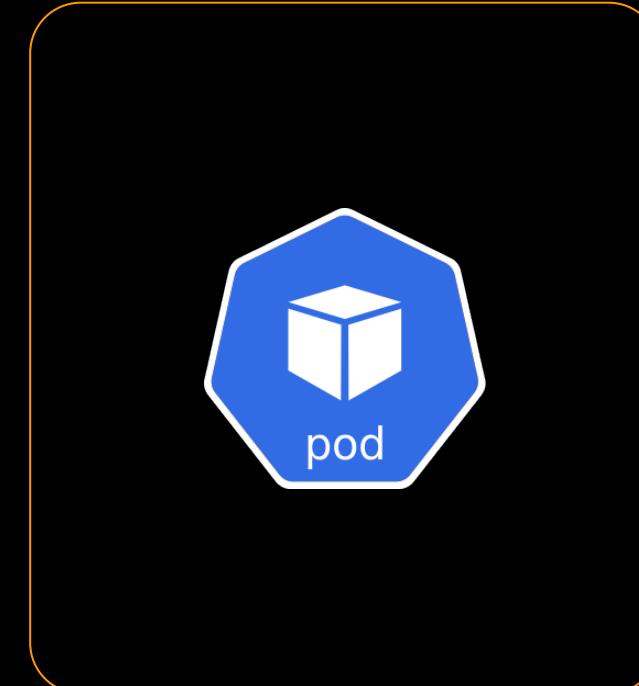
**template**

새 pod를 런칭하는데 사용할 템플릿. selectors의 값이 template의 labels과 일치해야 관리되는 파드를 제대로 선택.

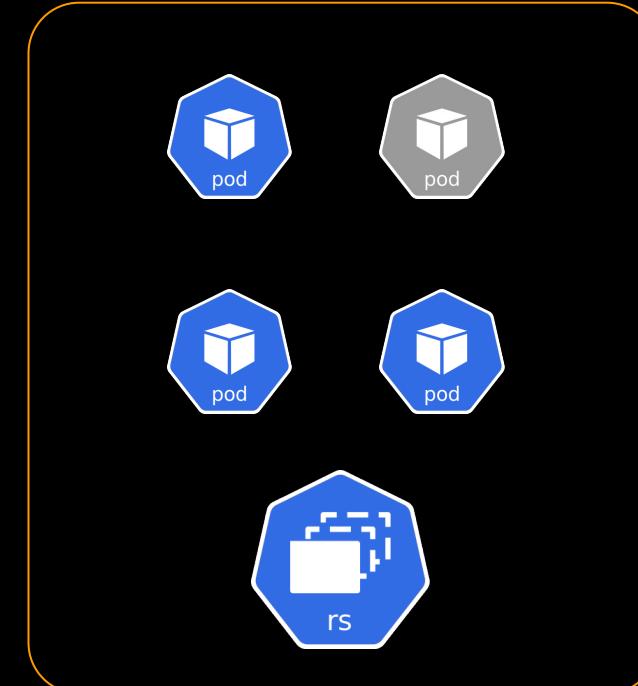
# Kubernetes Object



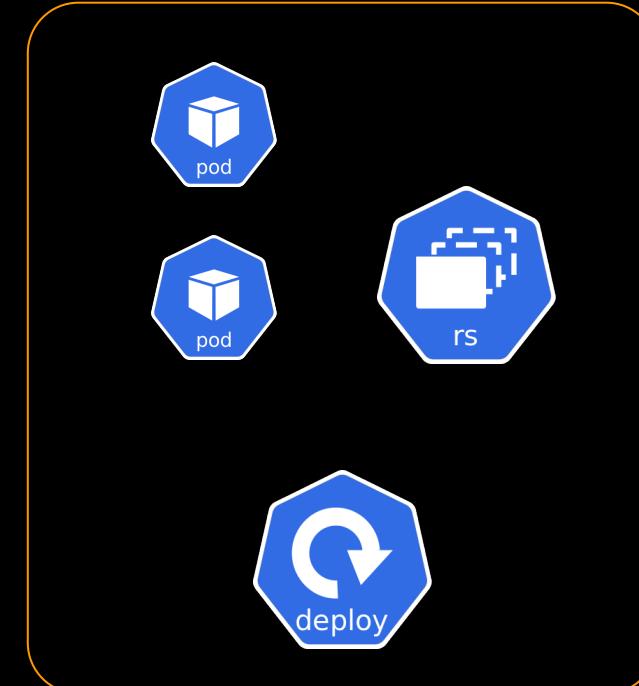
# Workload 생성을 위한 기본 컨셉



Pod



ReplicaSet



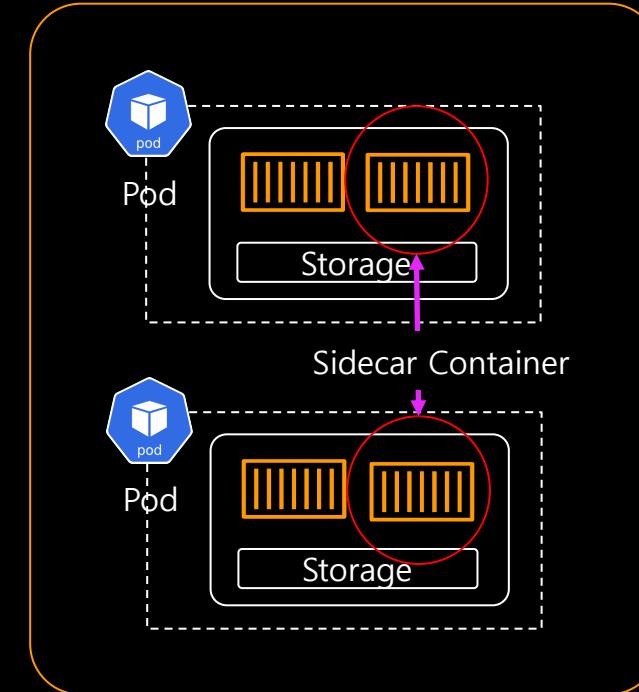
Deployment

Kubernetes 내에서 정의되는 가장 작은 단위의 워크로드 혹은 관리 리소스

Selector 를 기반으로하여 Pod 의 스케줄, 스케일링, 삭제를 담당

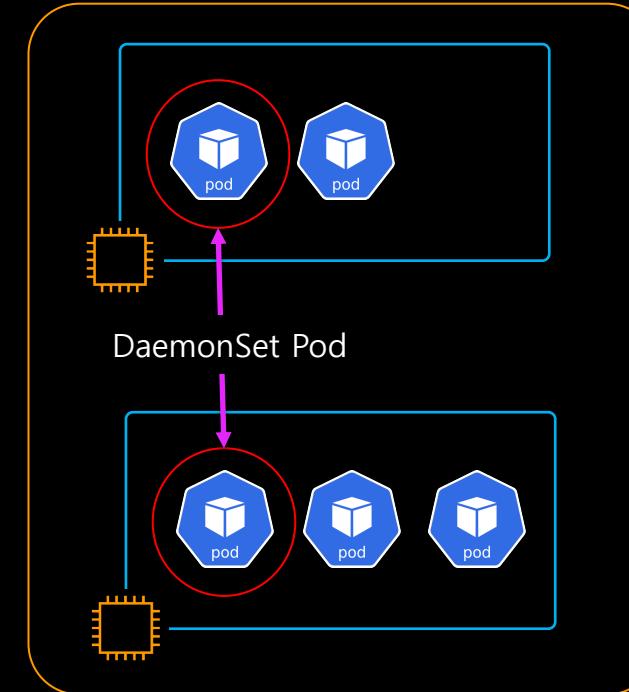
Pod 과 ReplicaSet 에 대한 선언적 관리 방법

# Workload 생성을 위한 기본 컨셉



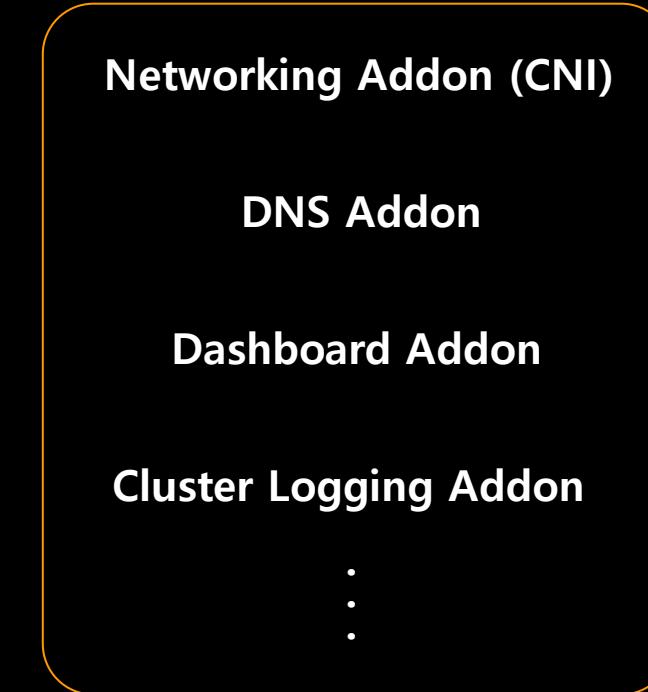
Sidecar

기본 컨테이너의 기능을 확장하거나 강화하는 용도의 컨테이너



DaemonSet

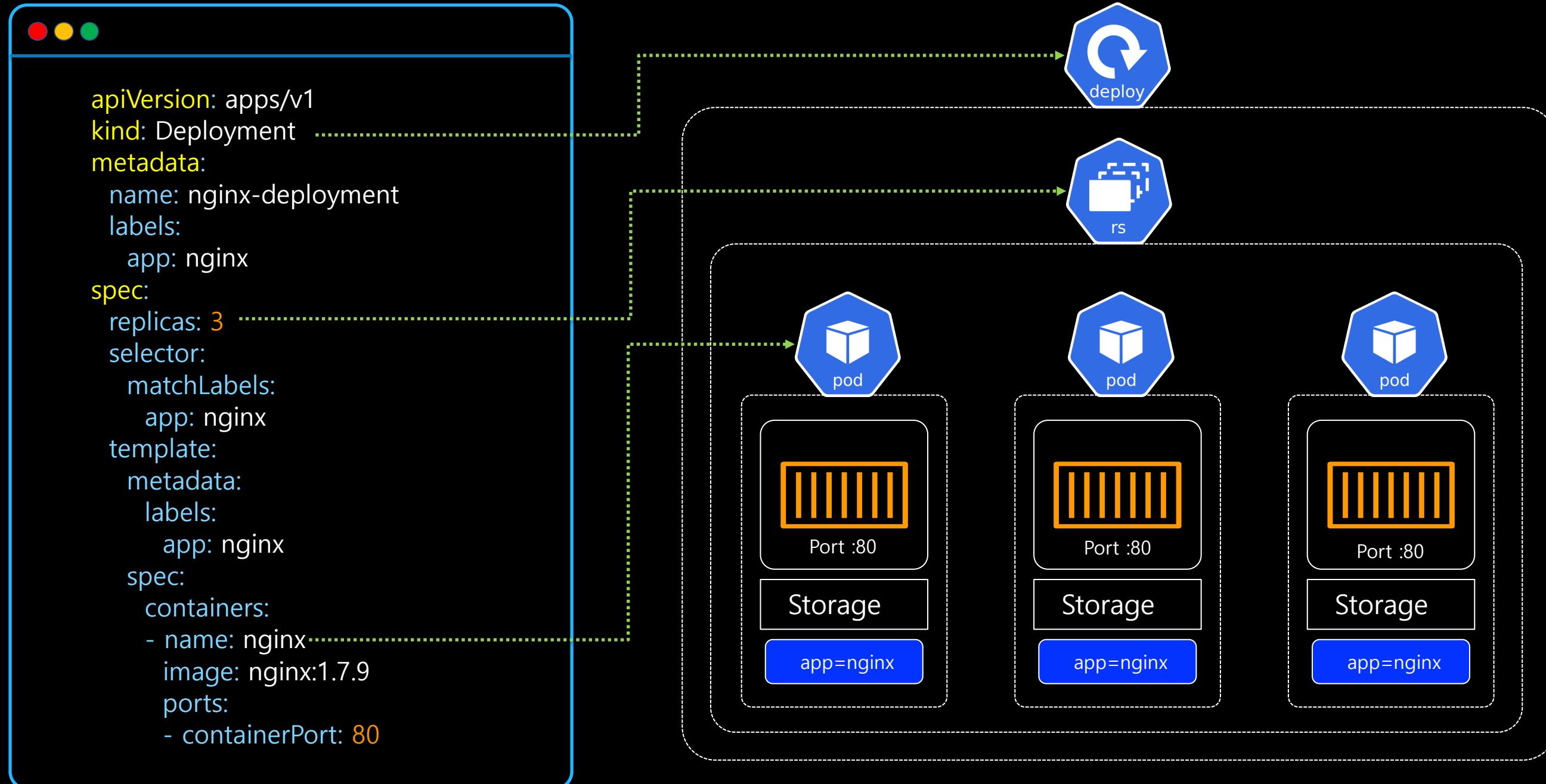
모든(또는 일부) 노드가 동일한 Pod를 실행



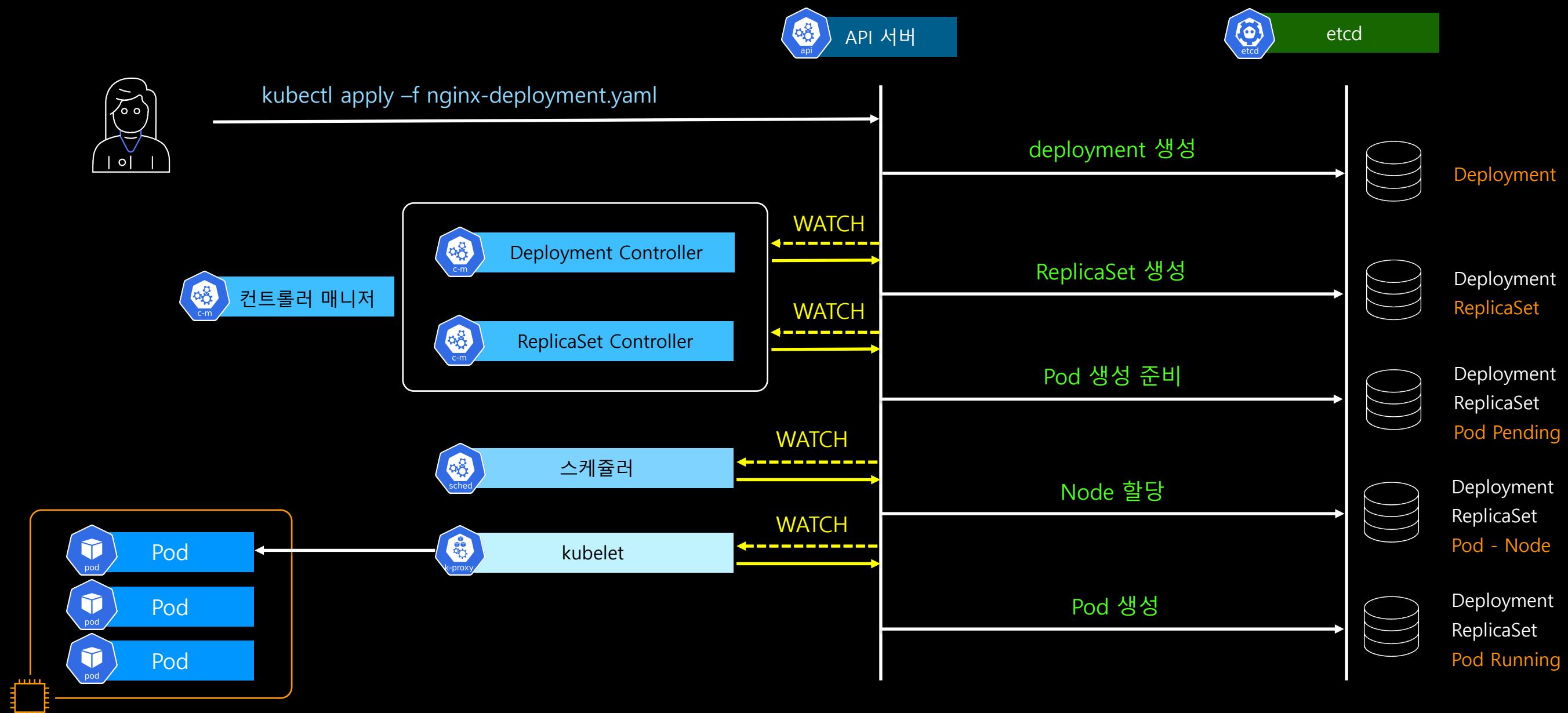
Addon

Kubernetes Cluster에서 기능을 구현 및 확장하는 역할

# Declarative Deployment

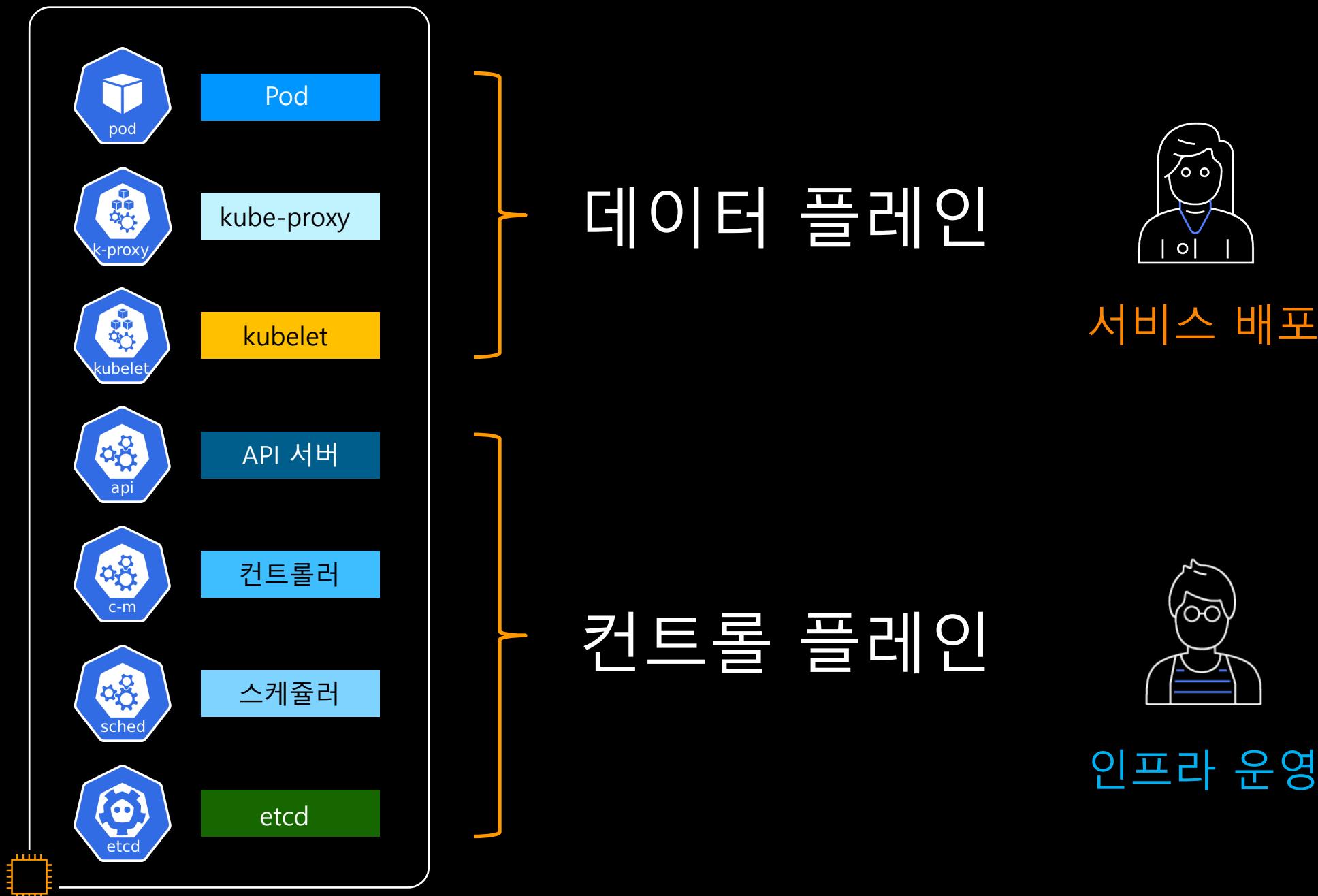


# K8S Pod 생성 흐름



# Clustering

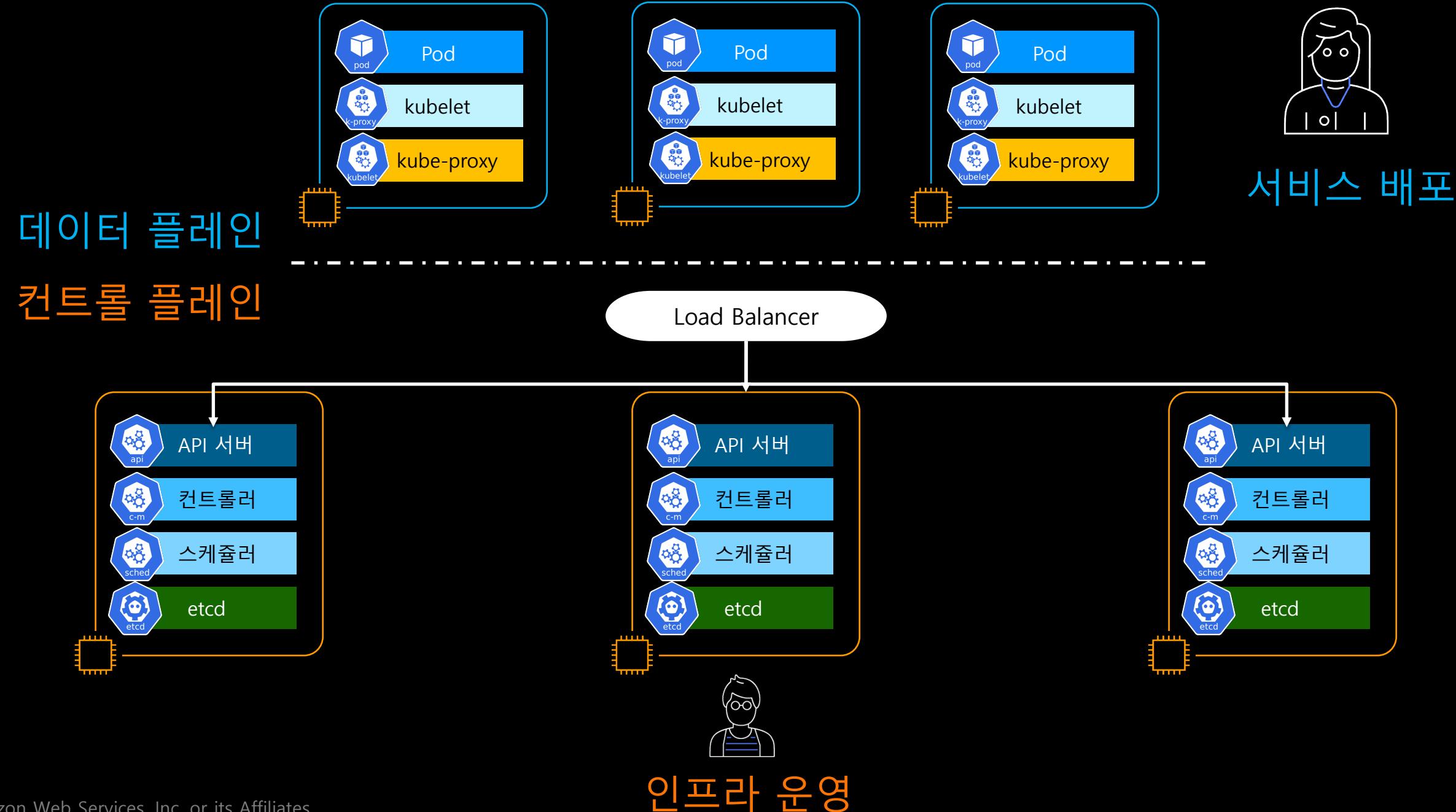
# Kubernetes - 단일 인스턴스



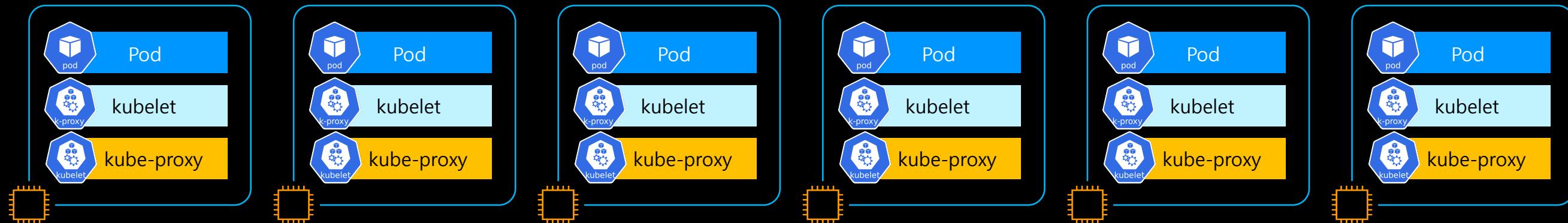
# Kubernetes – 다중 인스턴스



# Kubernetes – Stacked etcd



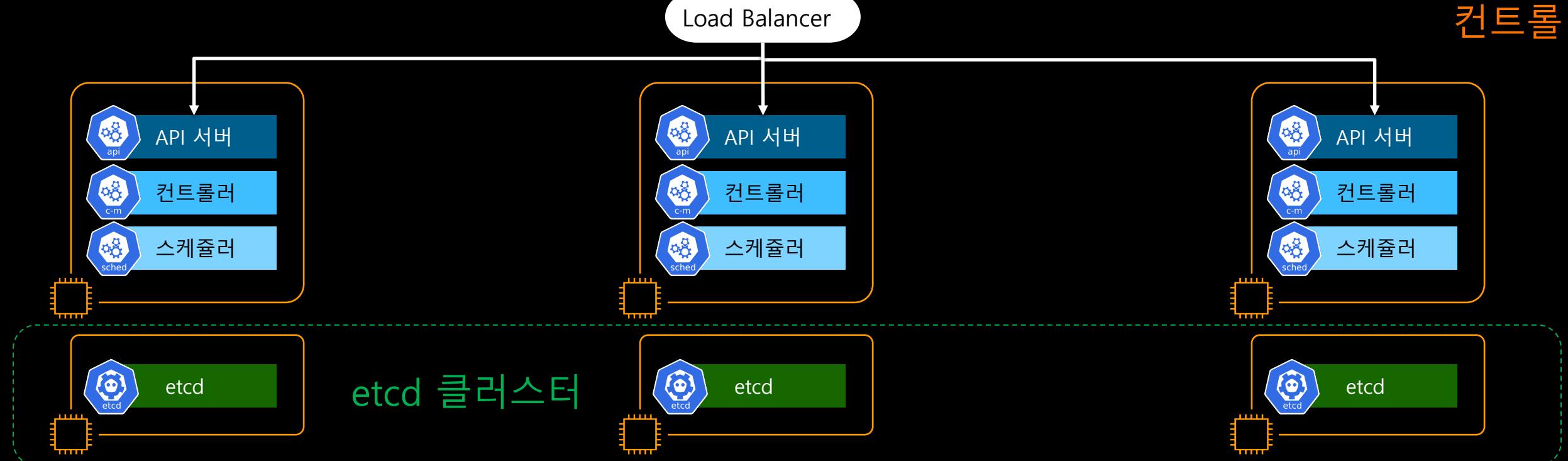
# Kubernetes – External etcd



데이터 플레인

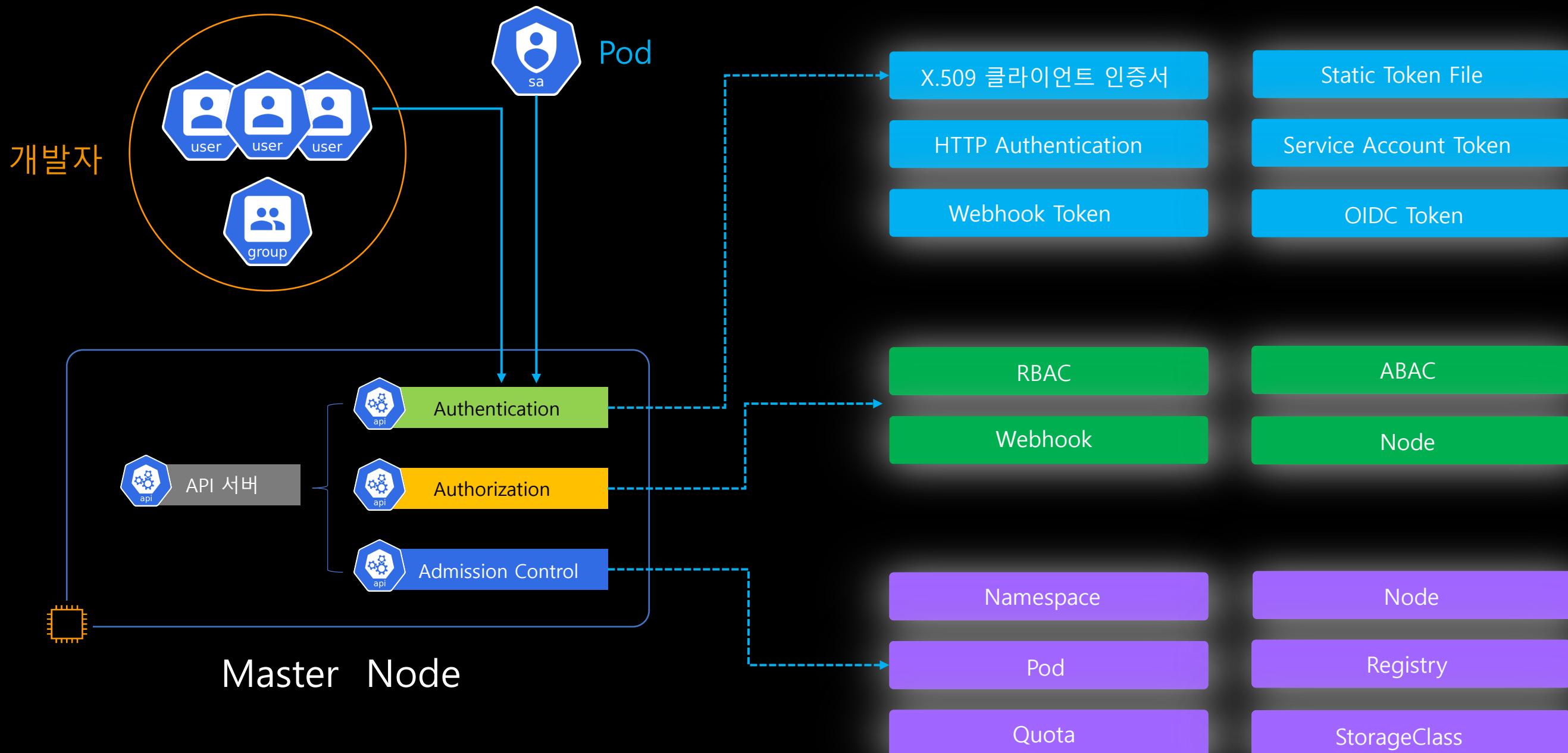
Load Balancer

컨트롤 플레인

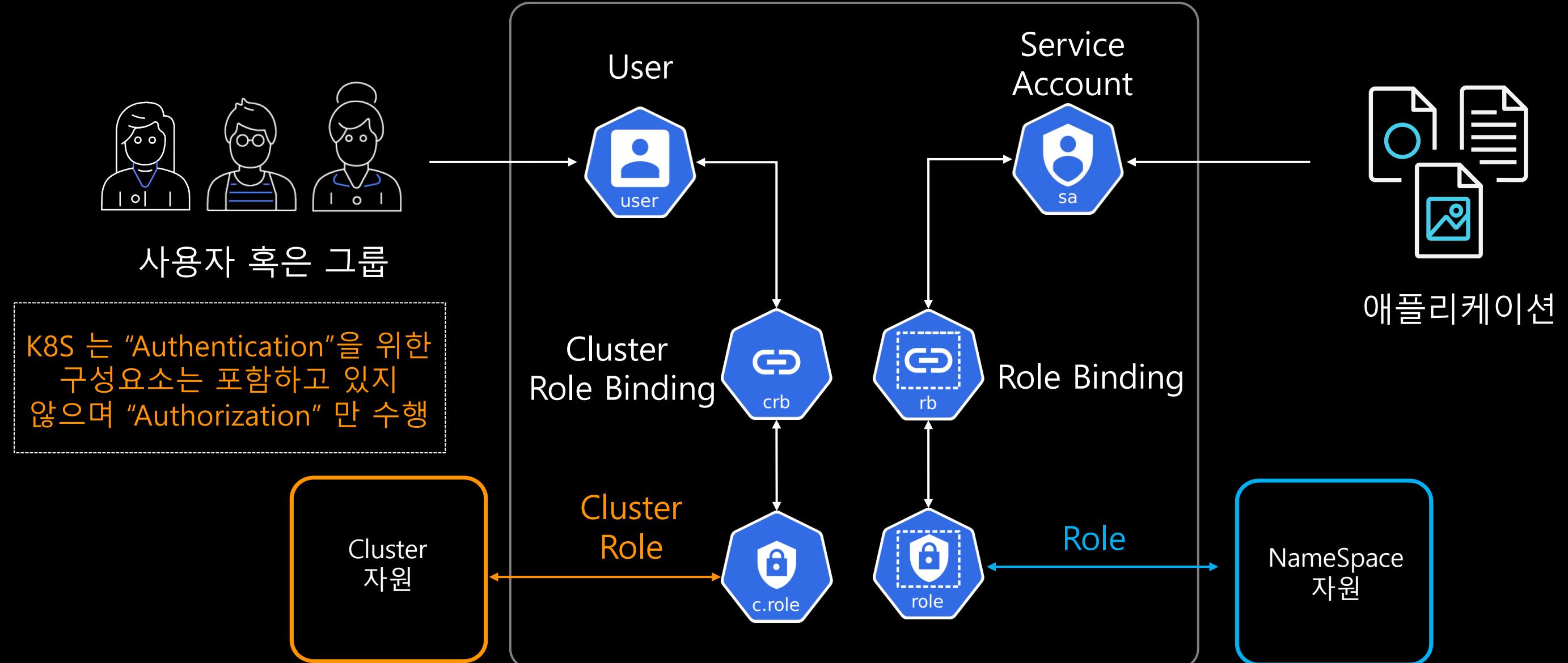


K8S 인증

# K8S 인증 및 인가

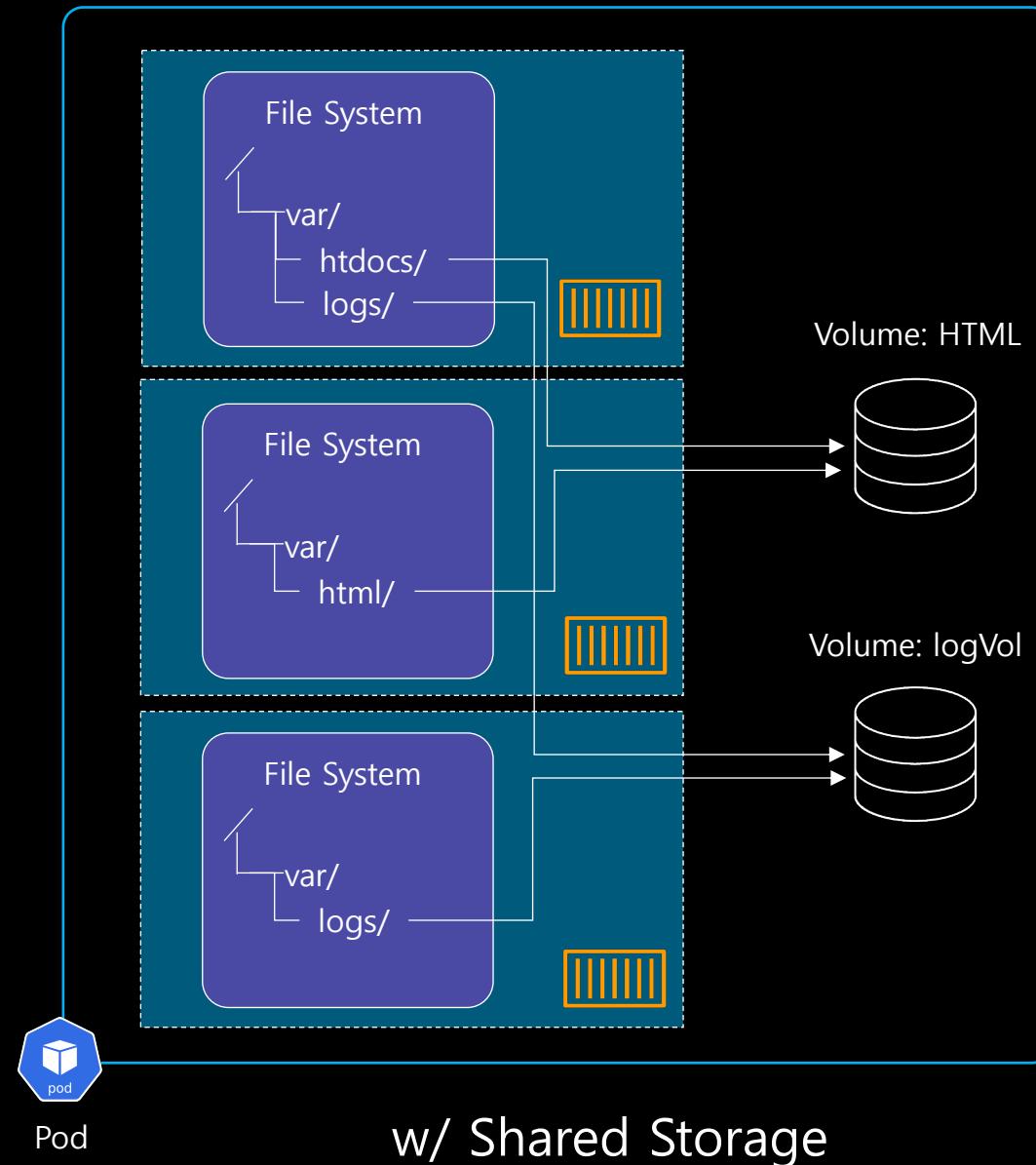
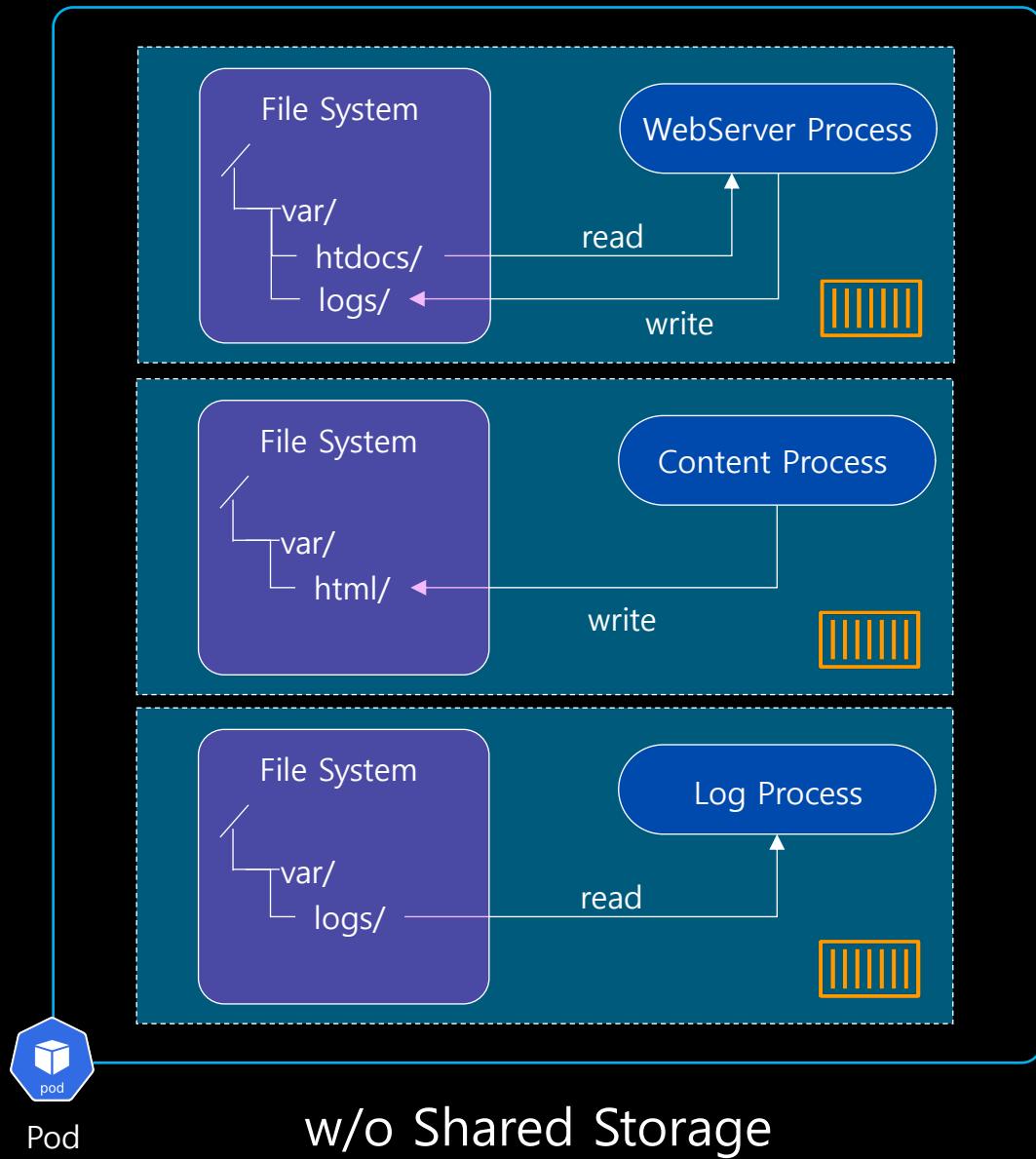


# K8S User & Service Account

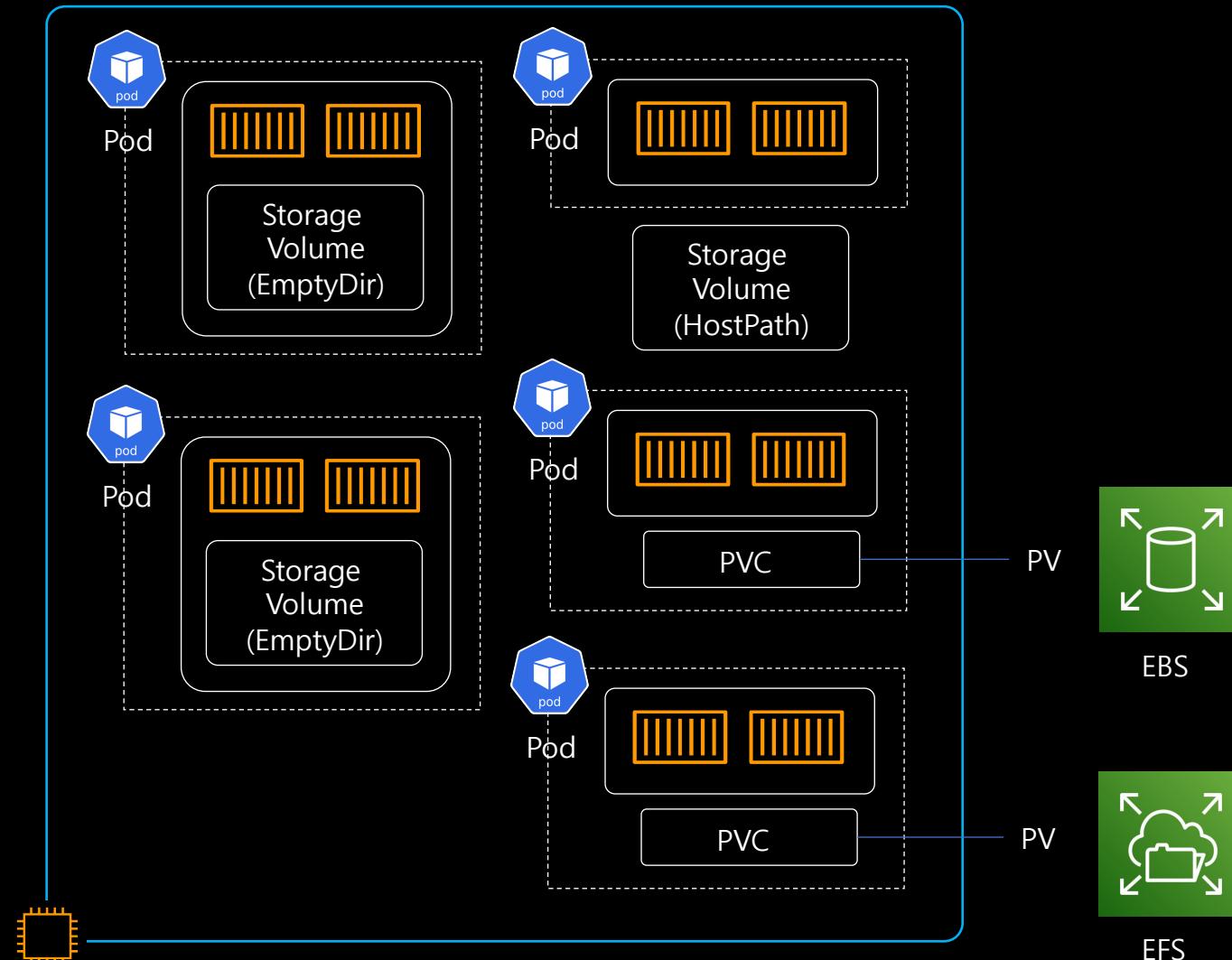


# K8S Storage

# Volume



# Volume



## EmptyDir

Pod가 실행되는 동안 사용되는 임시 저장 공간.

Pod 과 생명주기를 함께하는 휘발성 저장공간

## HostPath

Pod가 실행되는 Node 의 파일이나 경로를 마운트해서 사용

## Persistent Volume

Pod 와 독립적으로 존재하는 리소스

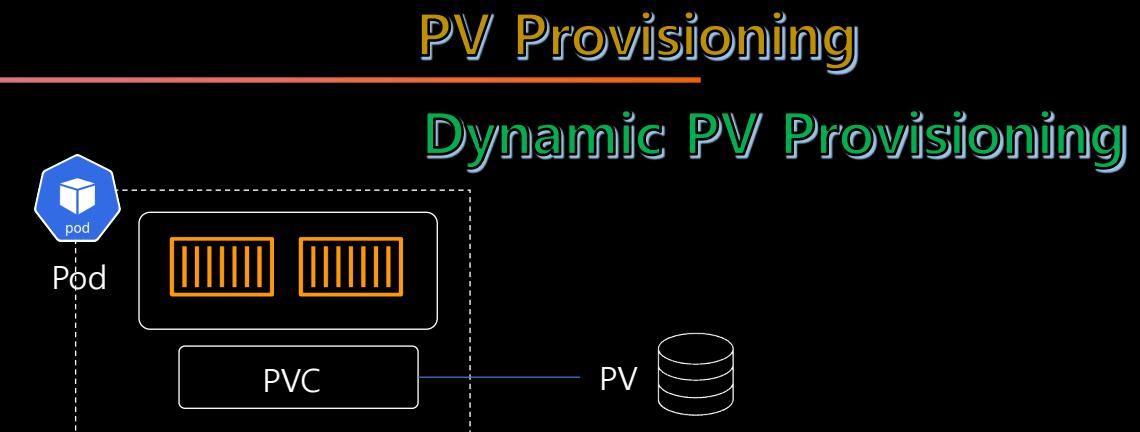
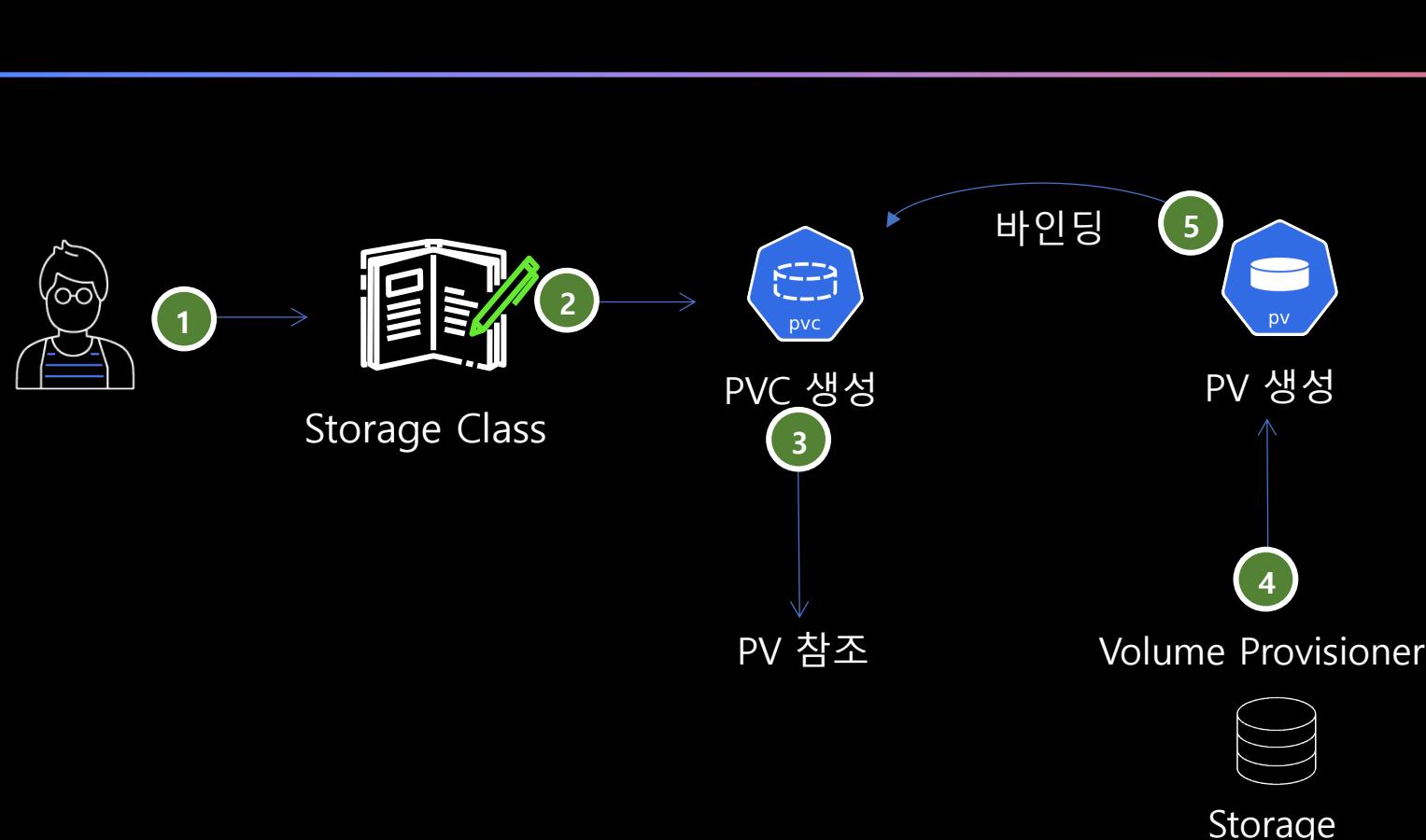
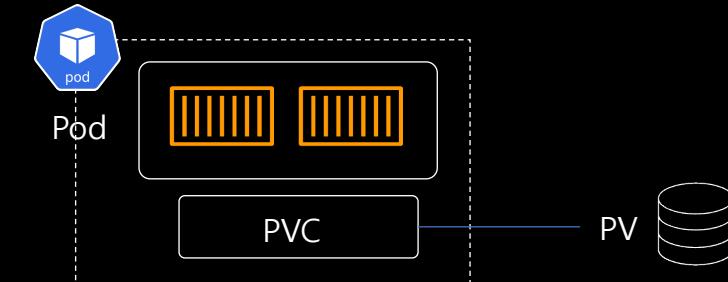
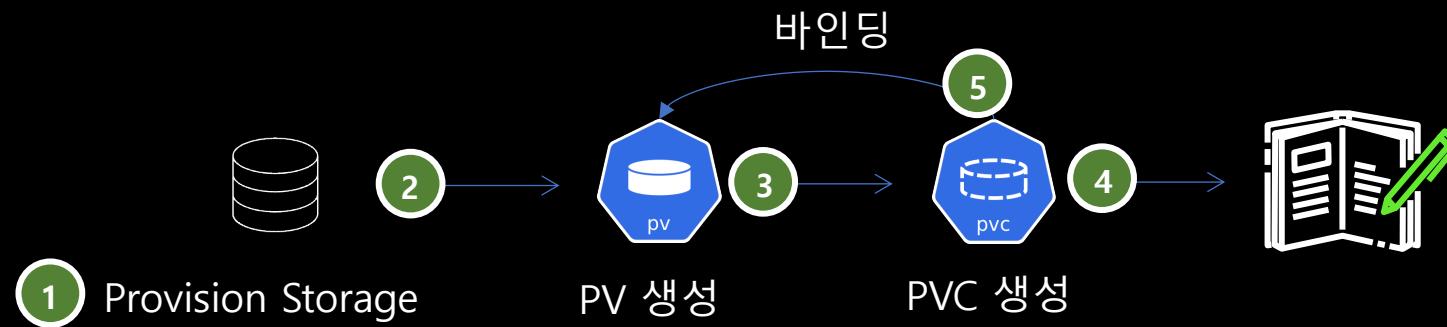
PVC 를 생성한 후 PV 를 연결하는 방식으로 사용

Storage Class 를 이용하여 동적으로 PV 를 할당 가능

## configMap, Secret, downwardAPI

특정 K8S 자원이나 정보를 Pod 에 노출하기 위한 특수 목적 Volume

# Persistent Volume 생명주기

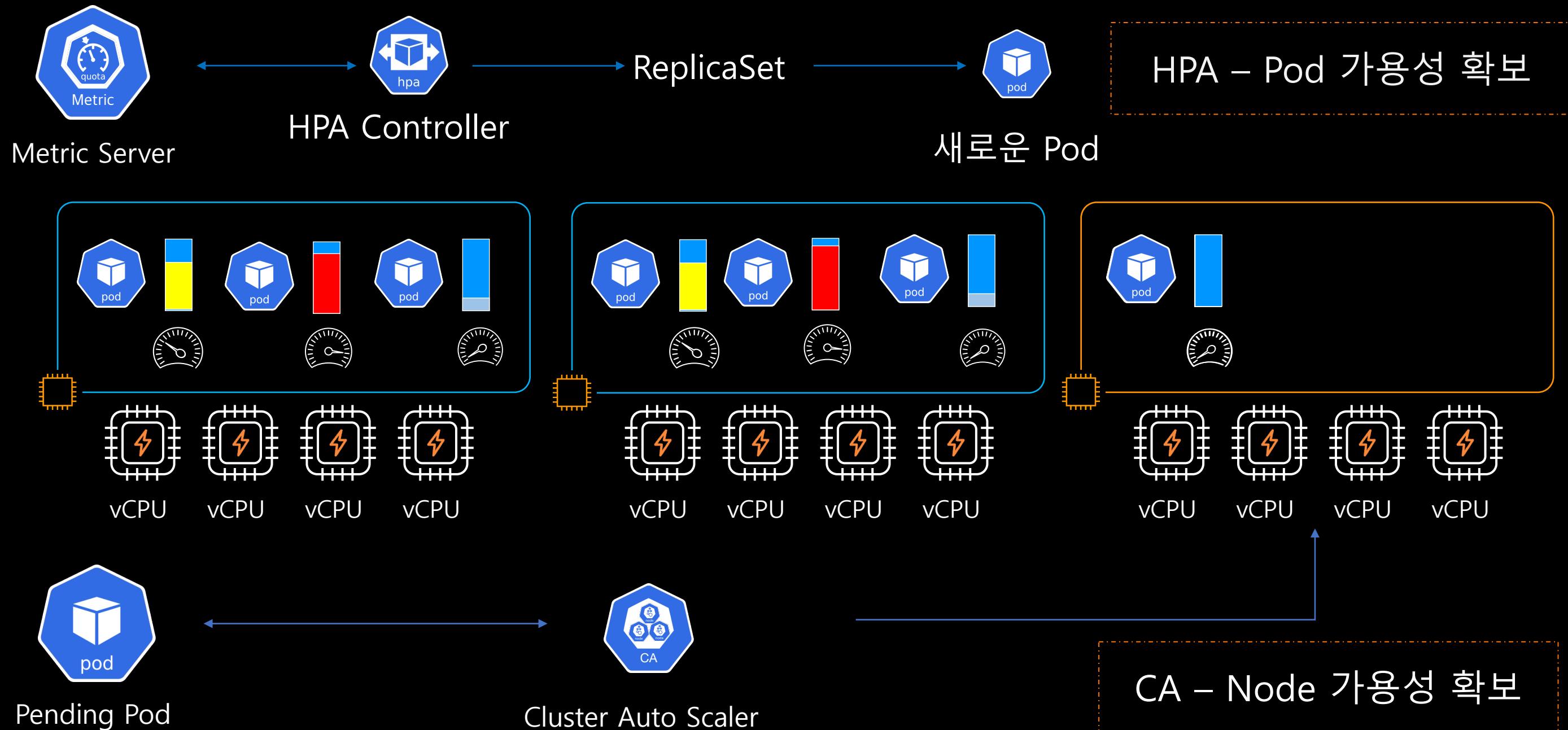


# Scaling

# Pod/Node 자동 확장

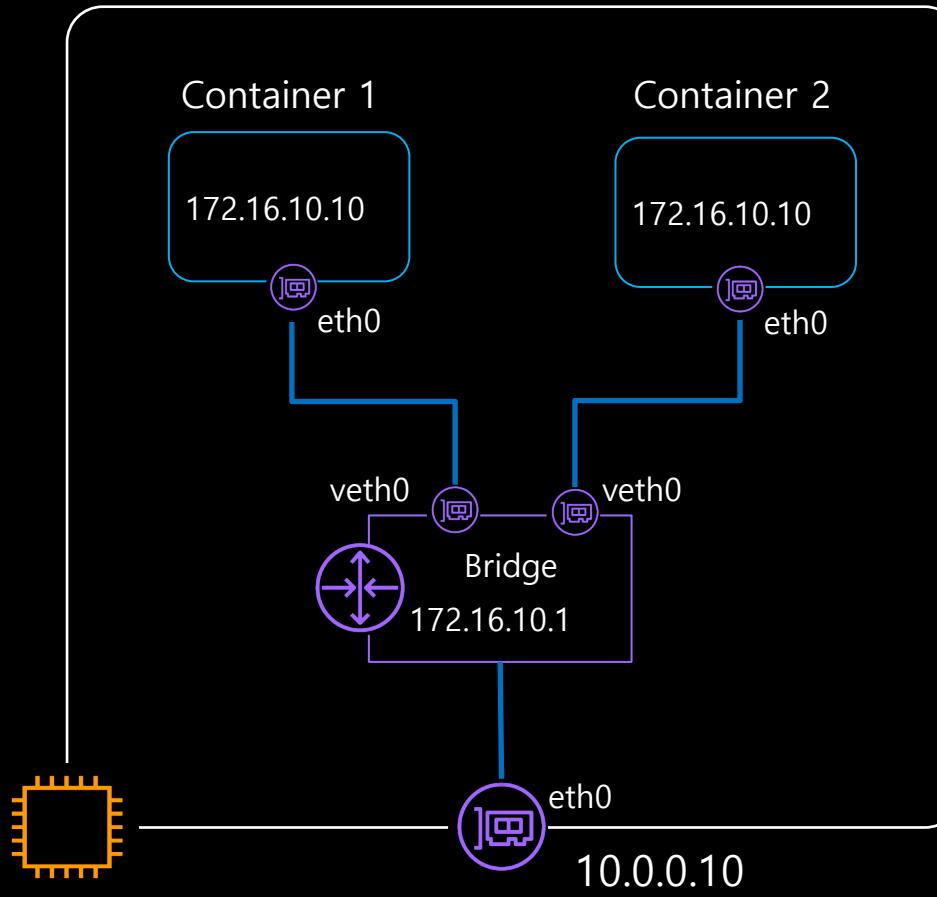


# Pod/Node 자동 확장



# K8S 네트워킹

# Container vs Pod 네트워킹



## Bridge

일반적인 Container 의 기본 네트워크 구조 – Bridge 를 통하여 Container 간 통신

## Host

Host 의 네트워크를 Container 와 공유하여 사용

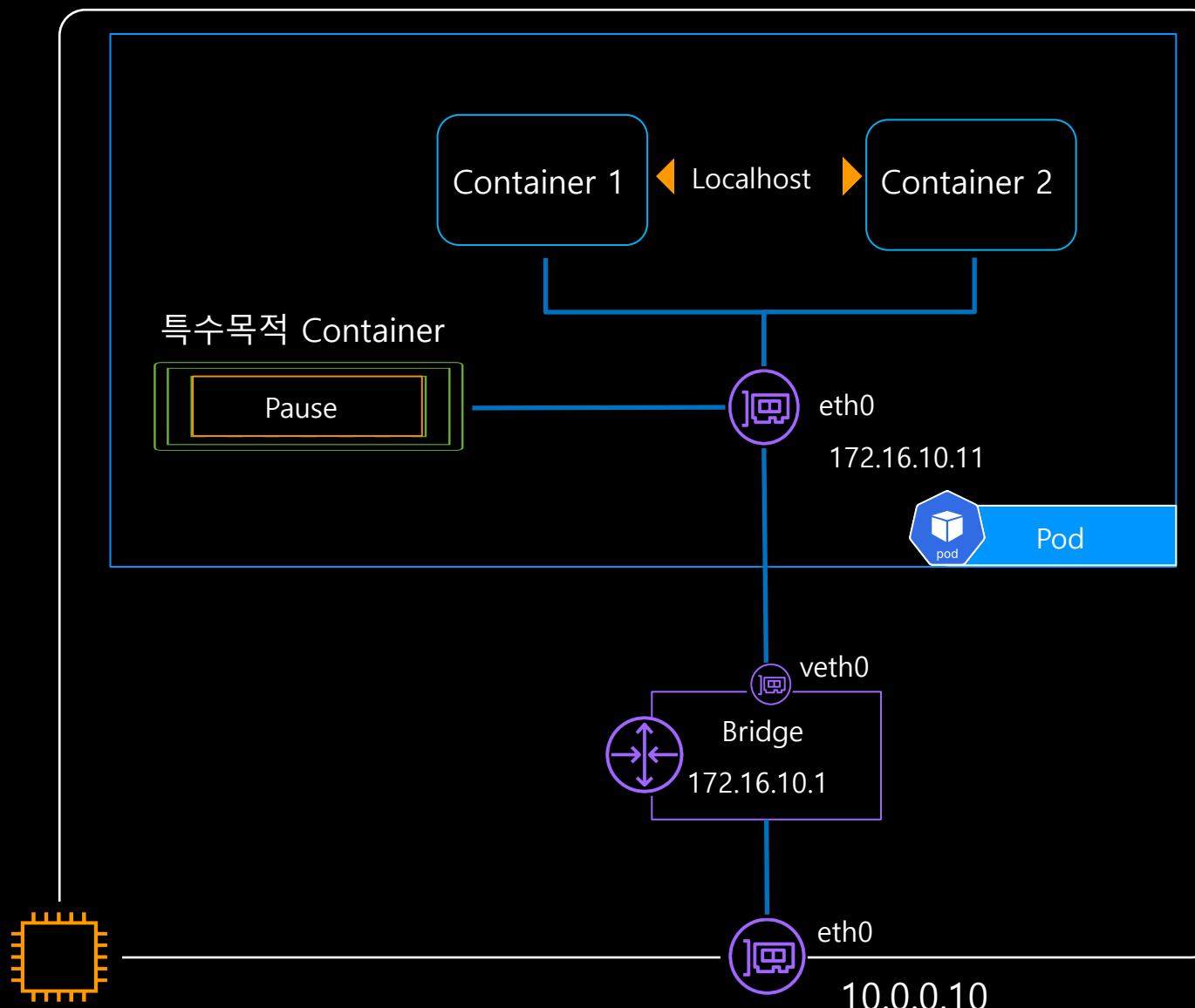
## Container

특정 Container 의 네트워크를 다른 Container 에서 공유하여 사용

## None

별도의 네트워크 인터페이스를 생성하지 않고 Local Network 만 사용

# Pod 네트워킹



## Network Name Space

Pause Container 의 Network Namespace 를 공유

## IP 주소

Pause Container 의 IP 주소를 공유받아서 사용

## Container 간 통신

Pod 내의 Container 는 Localhost 통신

## Pod 간 통신

동일 Node 에 위치한 경우 Bridge 를 통하여 통신

다른 Node 에 위치한 경우 CNI 이용

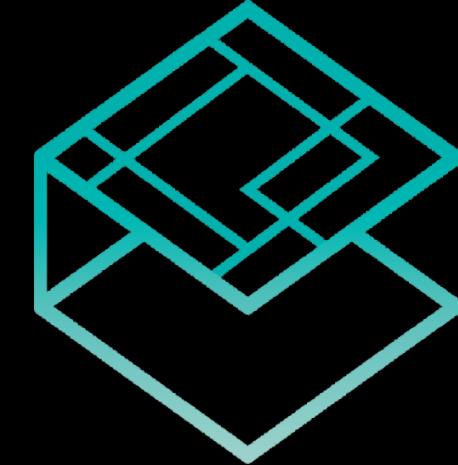
# Container Network Interface



Calico



Cilium

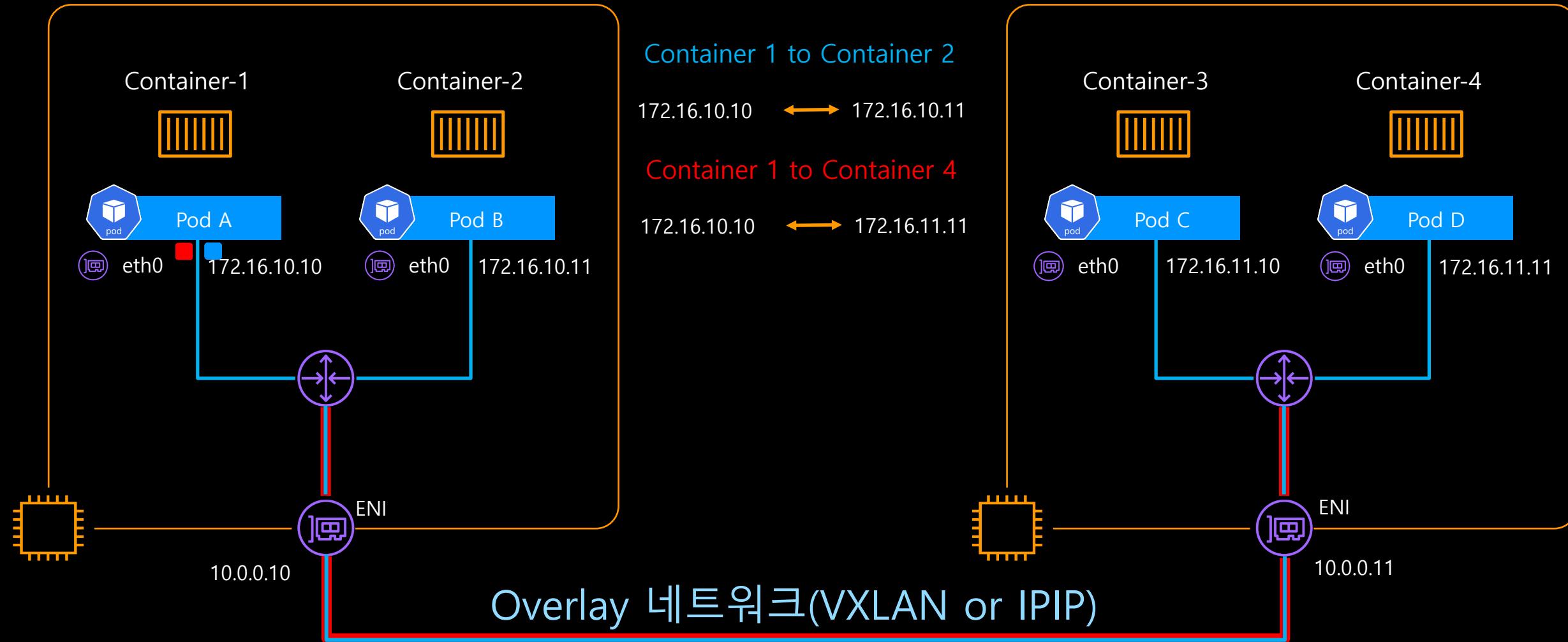


VPC CNI

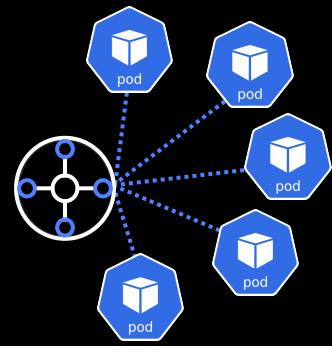
호환 가능한 Network Plugin

- Calico
- Cilium
- Contiv
- Contrail
- Flannel
- VPC
- kube-router
- Multus
- OpenVSwitch
- OVN
- Romana
- Weave

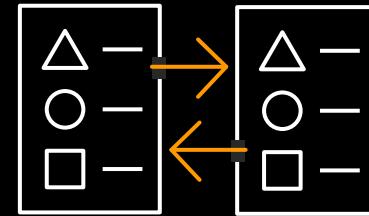
# Kubernetes 네트워킹



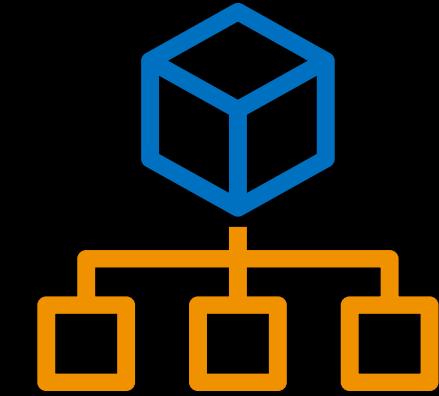
# K8S 서비스 타입



Cluster IP



Node Port



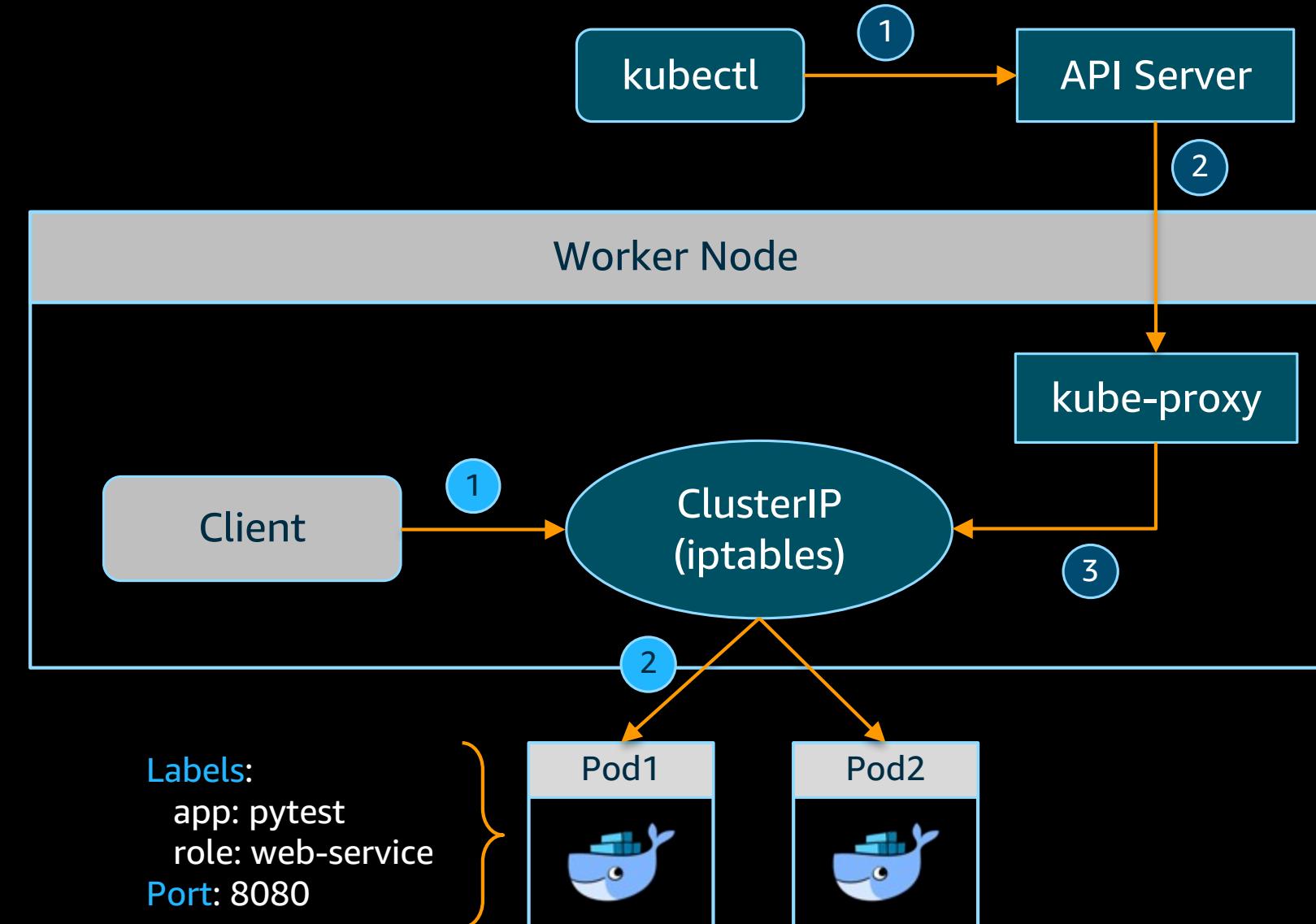
Load Balancer

# Traffic Routing from Service to Pods

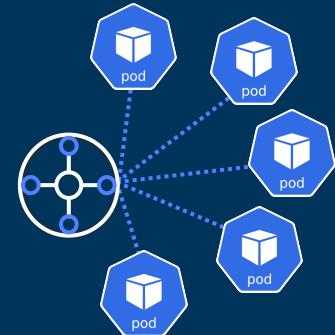
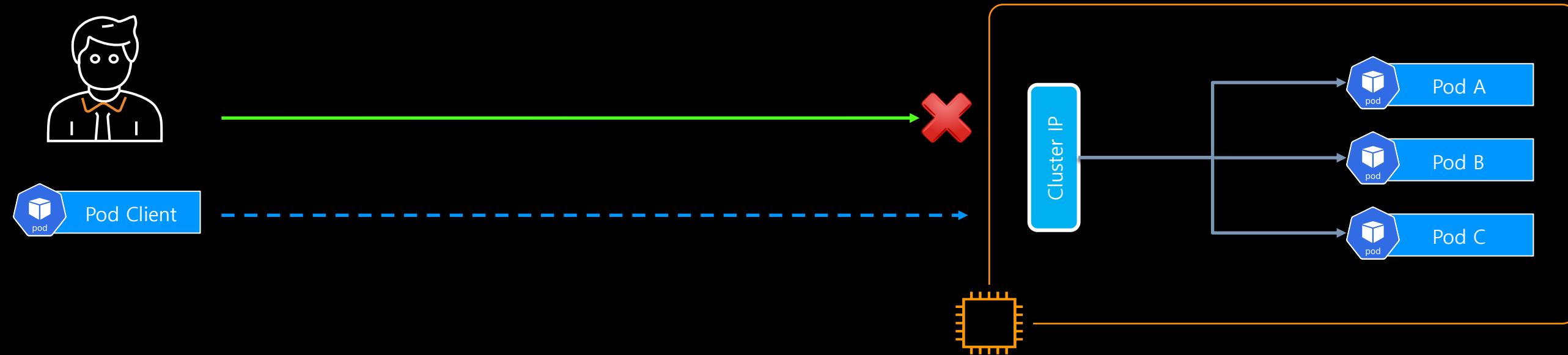
Service deployment does not create any running process that listens to incoming traffic

`kube-proxy` watches the Control Plane for addition and removal of a `Service`

Installs `iptables` rules, which redirect all traffic directed to the Service's address (cluster IP & port) to one of the backend Pods



# K8S서비스 탑입별 트래픽 흐름(Cluster IP)

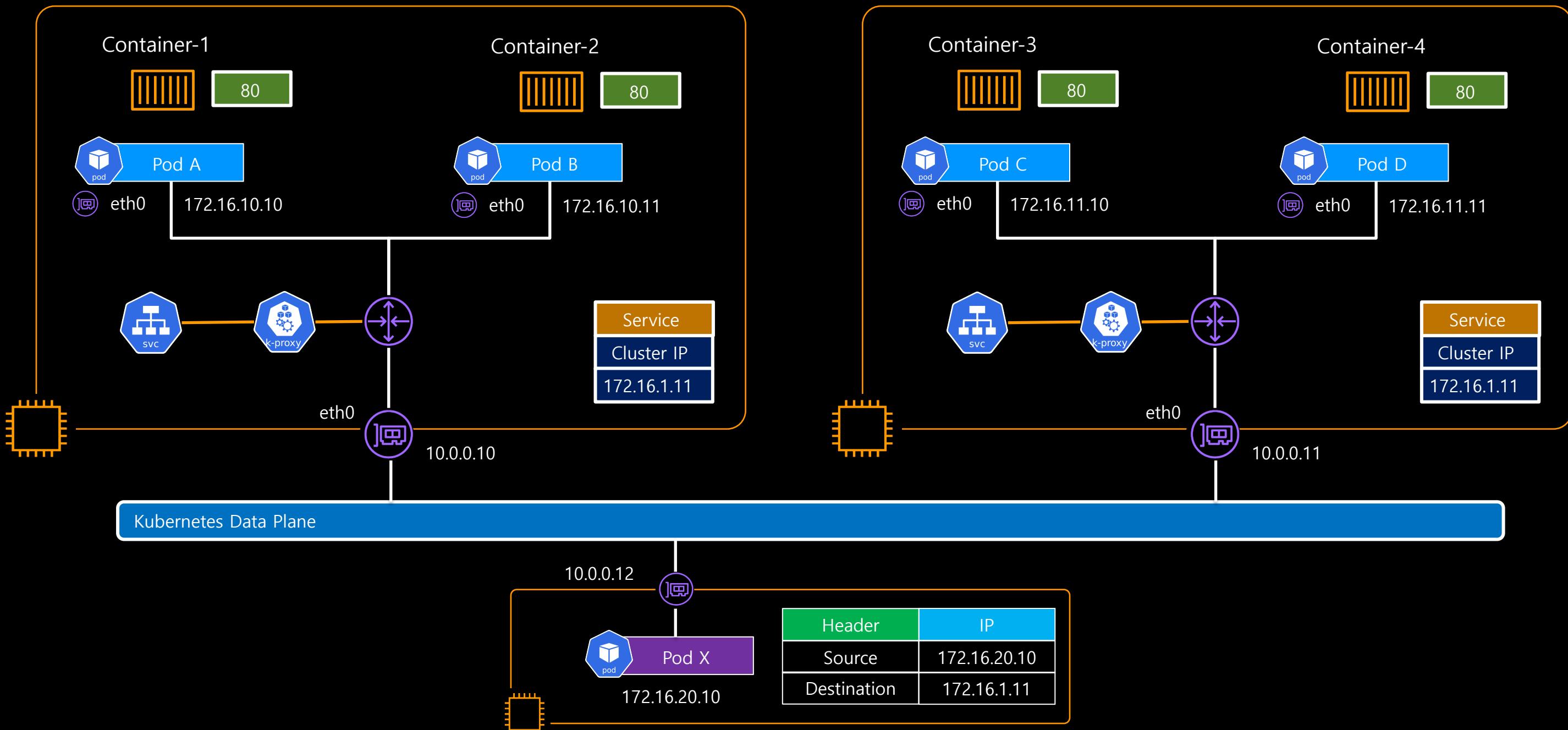


Cluster IP 는 Cluster 내부에서 서비스에 접속하고자 하는 용도로 사용

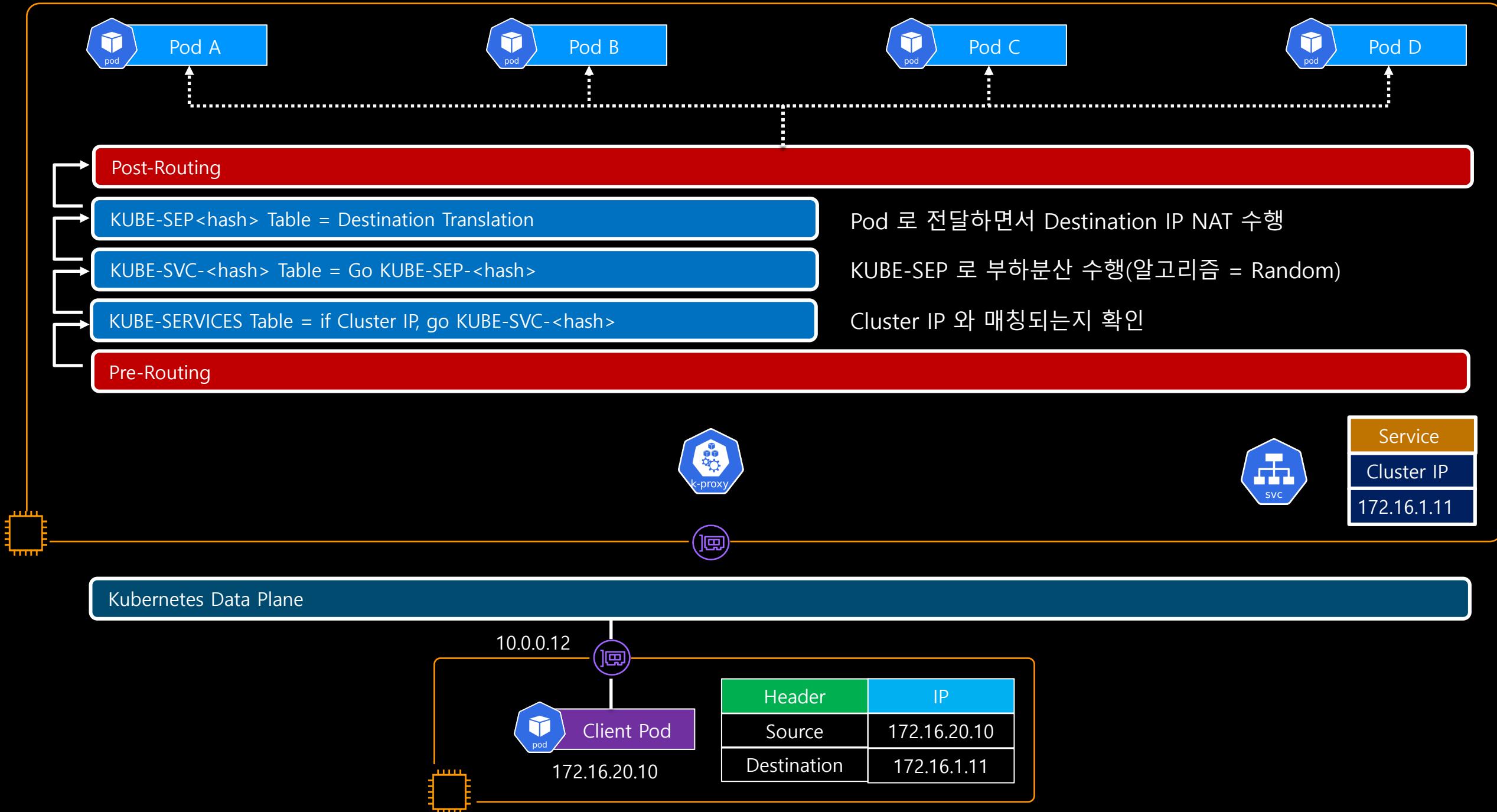
Cluster IP 에는 Pod 에 할당되지 않은 네트워크의 IP 주소를 할당

부하 분산에는 iptables(기본값)이나 IPVS(IP Virtual Server)를 선택적으로 사용

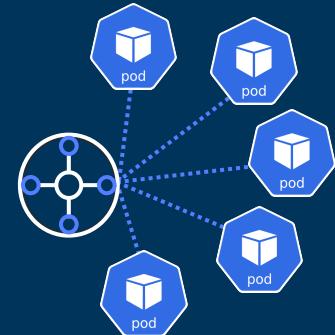
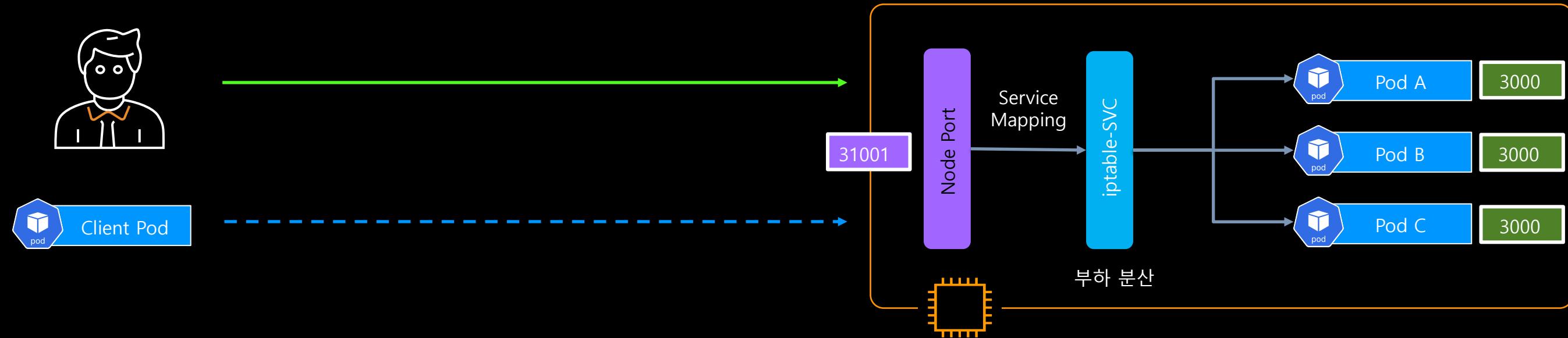
# K8S 서비스 네트워킹 – Cluster IP



# K8S 서비스 네트워킹 – Cluster IP(iptable mode)



# K8S서비스 탑입별 트래픽 흐름(NodePort)

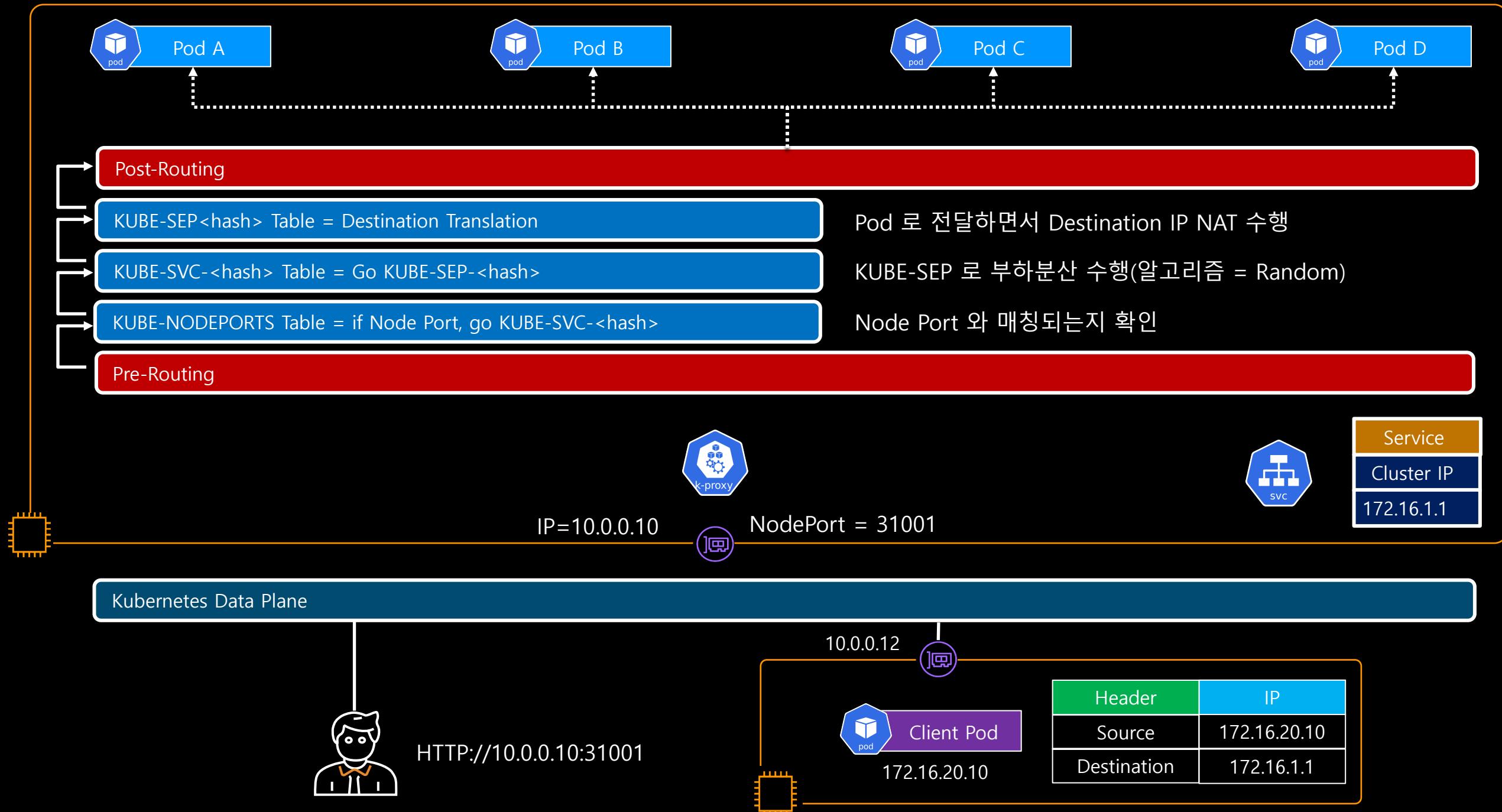


Worker Node 외부에 Port 를 Open 하여 내부 Pod 과 연결

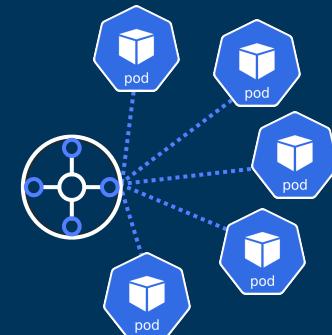
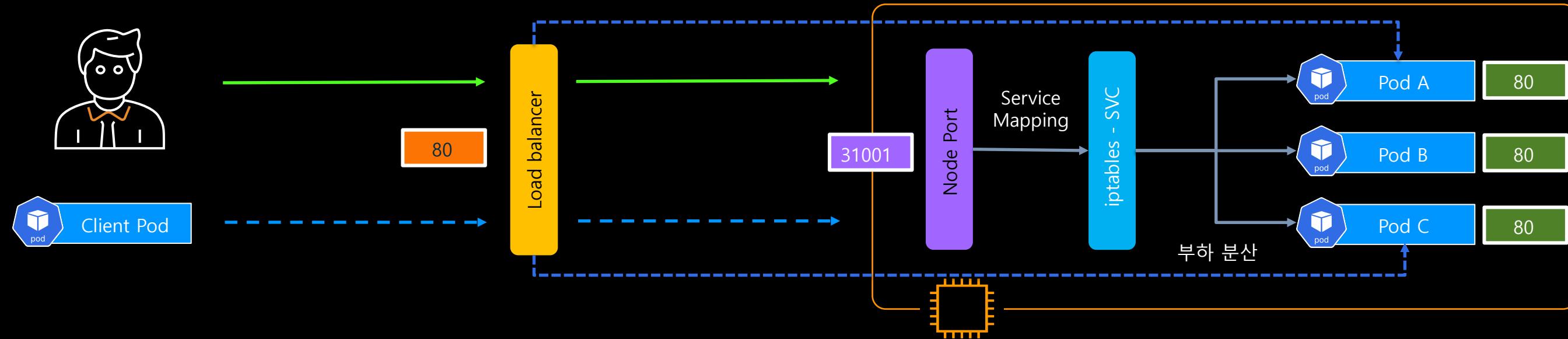
서로 다른 Node 간의 Pool 선택이 가능하며 필요 시 externalTrafficPolicy 를 Local 로 설정

Node Port 의 Port Range = 30000 ~ 32767

# K8S 서비스 네트워킹 – Node Port



# K8S서비스 탑별 트래픽 흐름(Load Balancer)

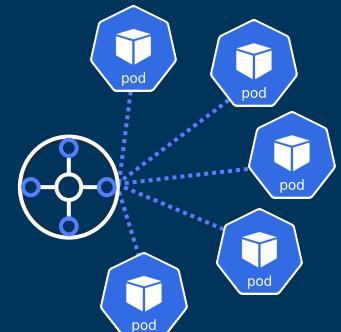
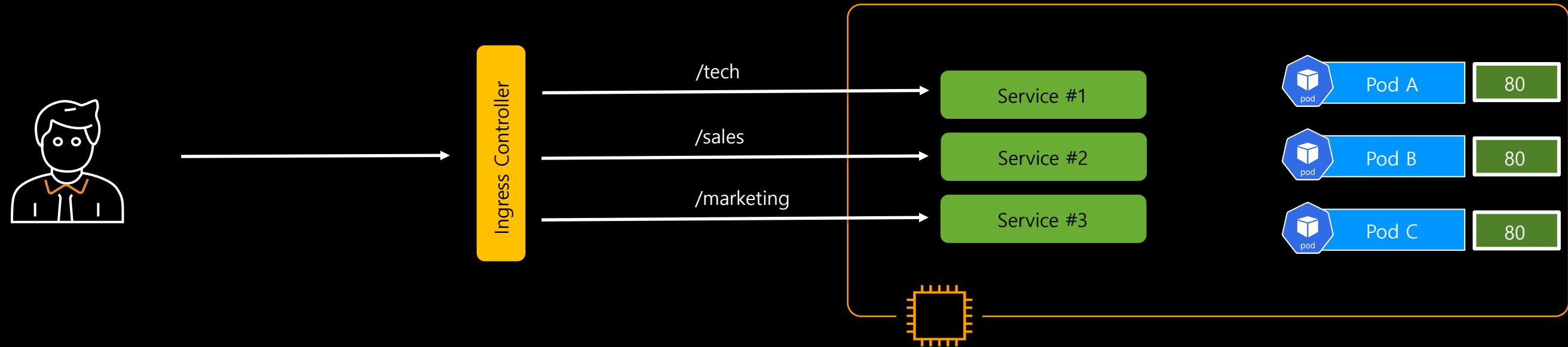


외부 접속용 Endpoint 역할을 수행할 Load Balancer 를 생성

외부 Load Balancer 는 각 Node 의 Node Port 를 Target 으로 등록(ALB/NLB 의 경우 IP 등록 지원)

Pod 의 부하 분산은 iptables에서 수행(Load Balancer 의 제공 기능에 따라 직접 분산도 가능)

# K8S서비스 탑입별 트래픽 흐름(Ingress)

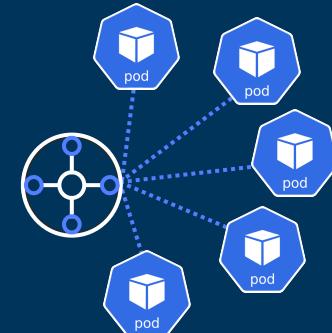
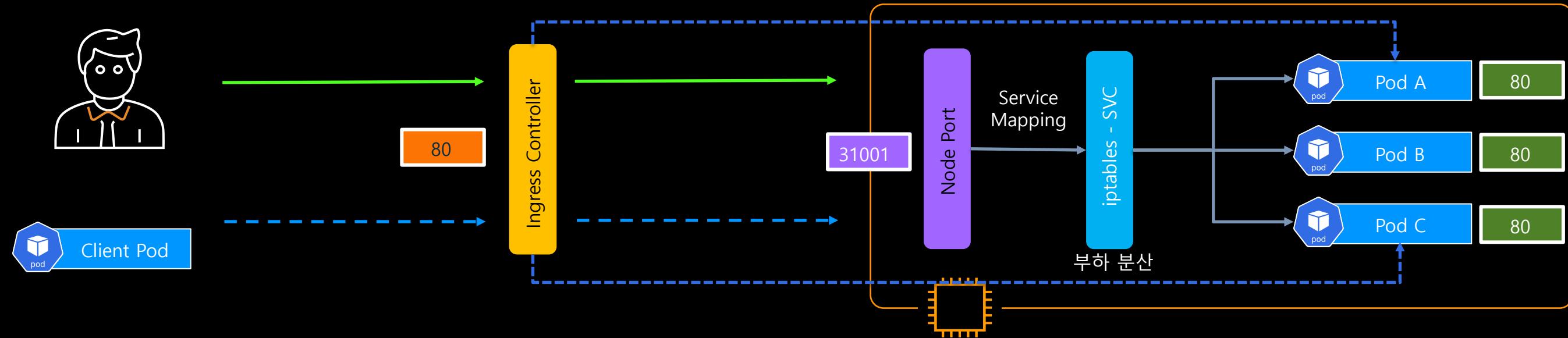


L7 영역의 관리자 정의 규칙을 기반으로 한 서비스 공개가 필요한 경우 사용

Ingress 를 통해 L7 영역에 대한 트래픽처리 규칙을 정의하고 Ingress Controller 를 통해 처리

Ingress Controller 는 NIGNX Controller 나 CSP 에서 제공하는 Controller 등 선택 가능

# K8S서비스 탑입별 트래픽 흐름(Ingress – EKS)



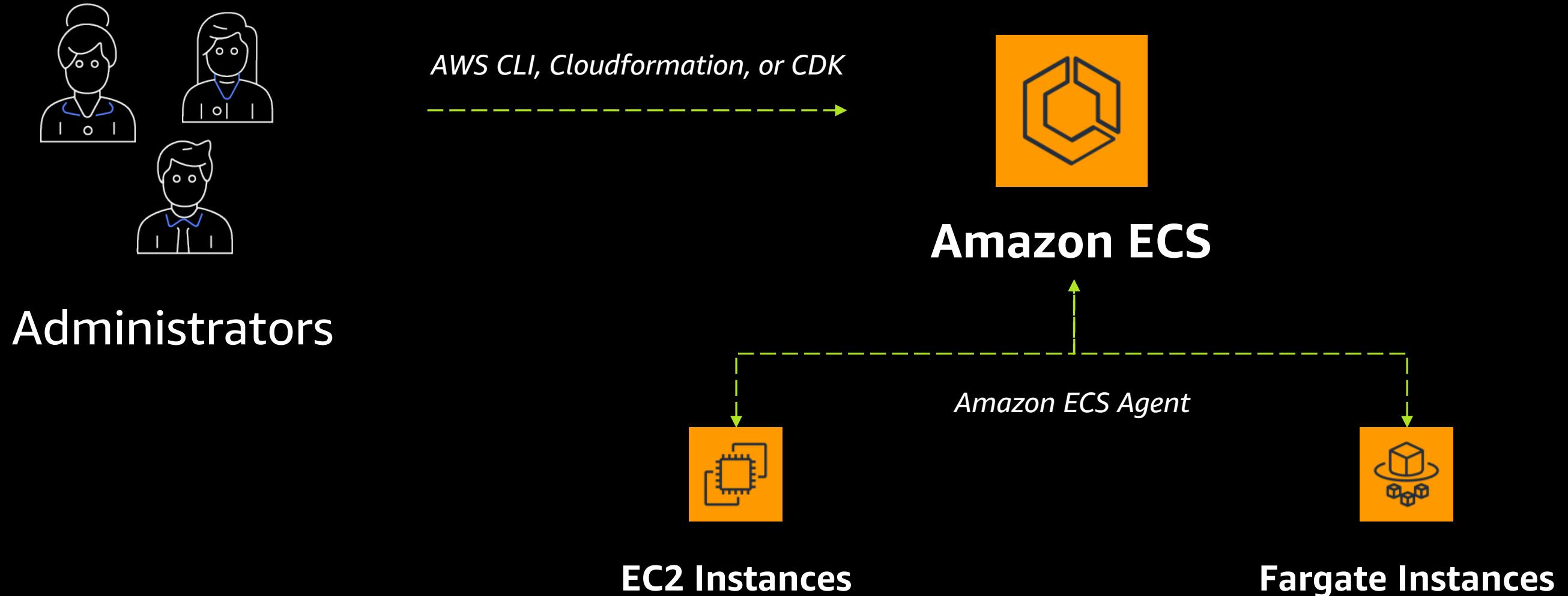
외부 Endpoint 역할을 하는 Ingress Controller(ALB)를 자동으로 생성

ALB 의 Target 은 Instance Mode 혹은 IP Mode 로 등록 가능(Sticky 필요 시 IP Mode 사용)

ALB 생성을 위하여 ELB 권한을 갖는 IAM Role 생성 및 연결 필요

# Amazon ECS

# Amazon ECS Architecture



# Amazon ECS 구성 요소



ECS 클러스터

작업이 실행되는 논리적 그룹



ECS 작업

실제 컨테이너 작업(최소 실행 단위)  
하나 혹은 둘 이상의 컨테이너의 묶음

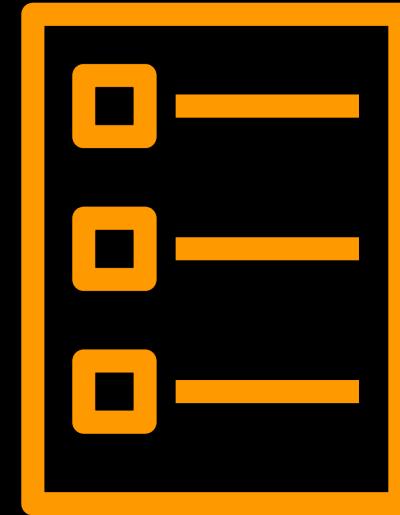


ECS 서비스

ECS 위에서 작업을 실행하는 방법  
정의한 작업 수를 유지하기 위해 지속적으로  
관리

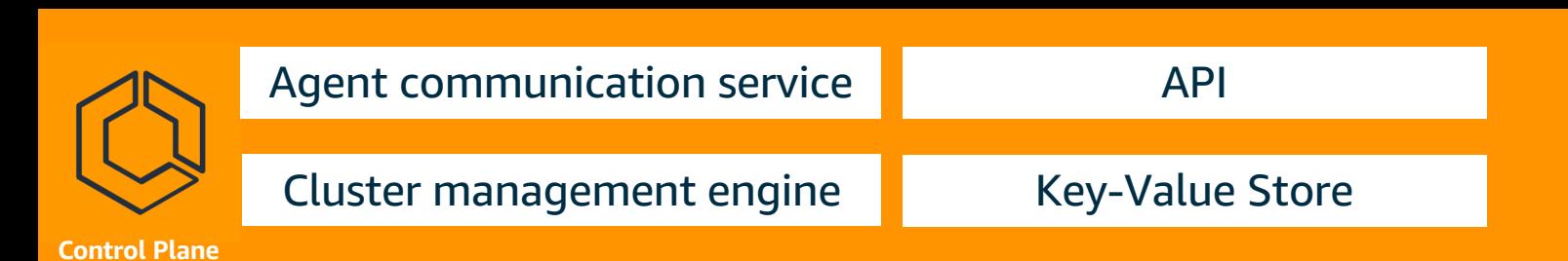
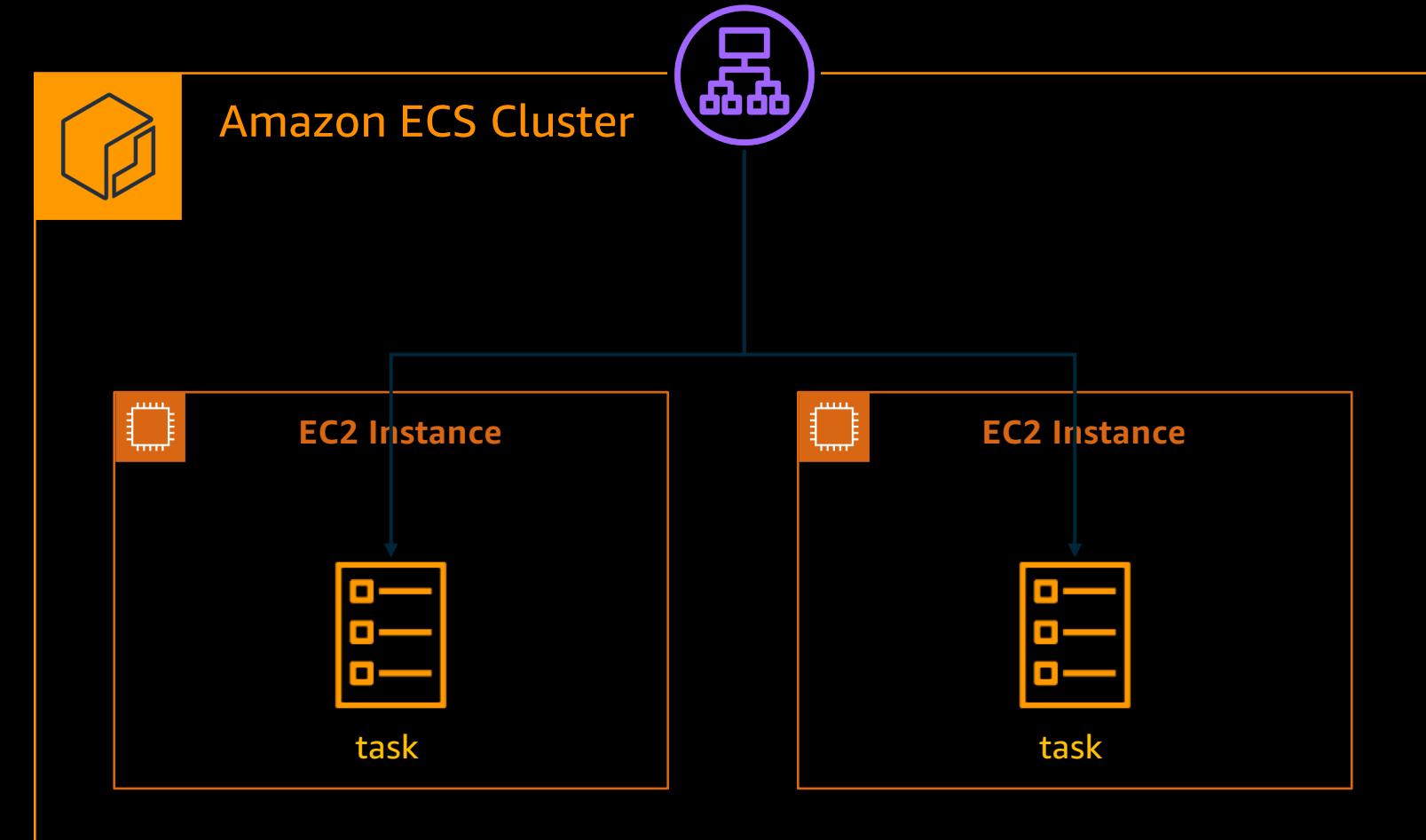
# Amazon ECS - 작업

- ECS 클러스터에서 최소 실행 단위
  - 네트워킹, 스토리지, 파라미터, IAM 역할, 컴퓨팅 리소스 등의 구성 값 설정 가능
- **작업 정의 (Task Definition)** 내용을 기반으로 작업 배포
  - 배포 타입 설정: Fargate, EC2
  - 작업 역할을 부여해 API 요청을 받을 때 권한에 따라 동작
  - 작업 크기 설정

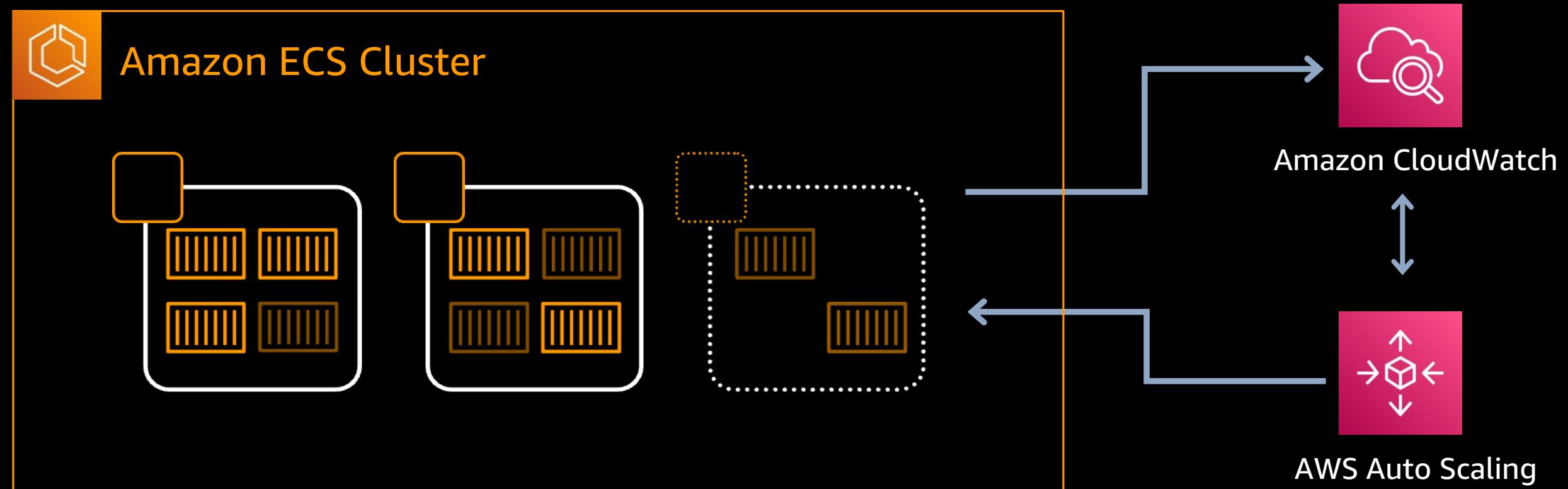


# Amazon ECS - 서비스

- 설정한 작업 수를 자동 유지 및 복구
- 서비스 유형 (replica, daemon) 선택
- 롤링 업데이트, 블루/그린 배포
- AWS ELB를 통한 효율적 부하 분산
- 서비스 Auto Scaling 적용 가능



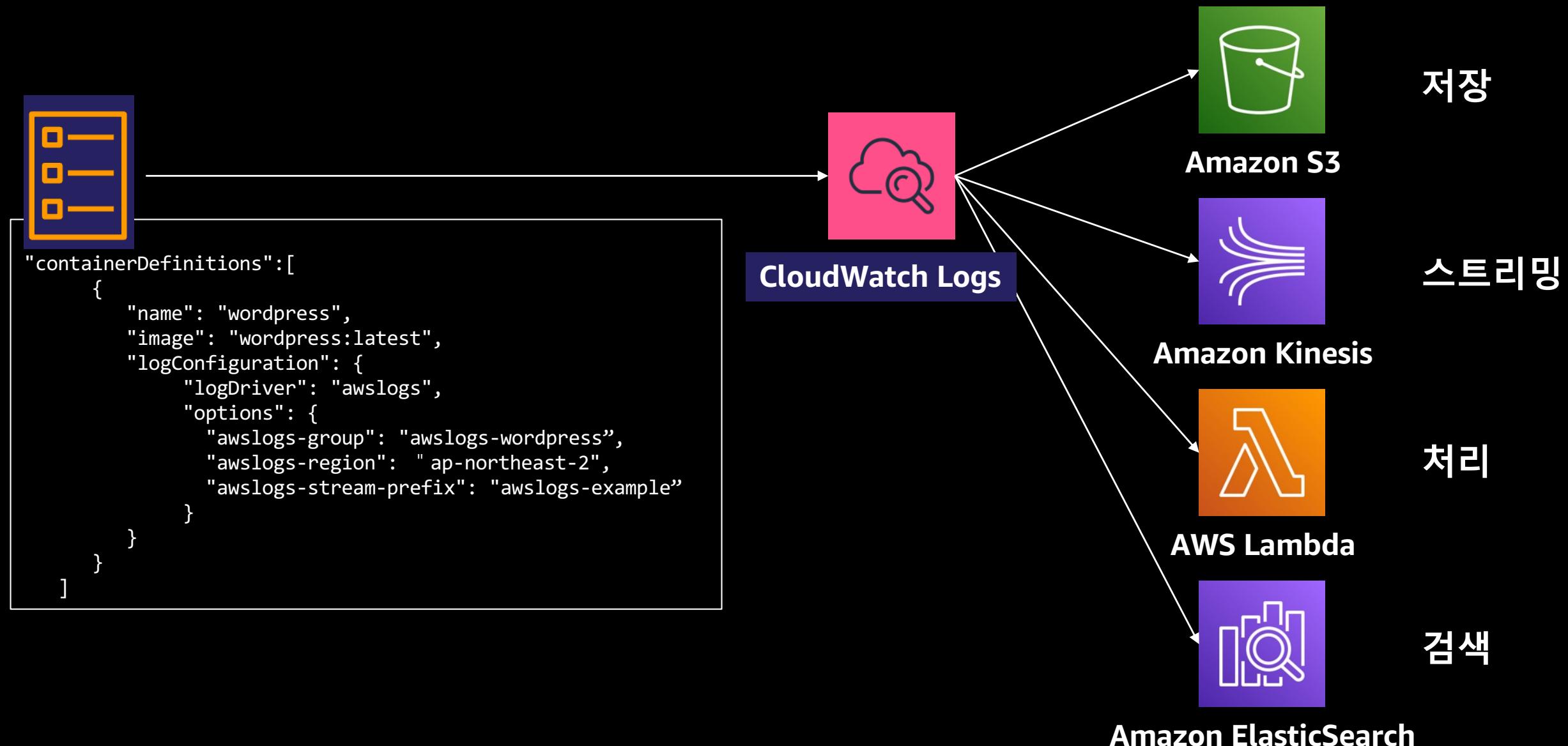
# ECS에서의 Auto Scaling



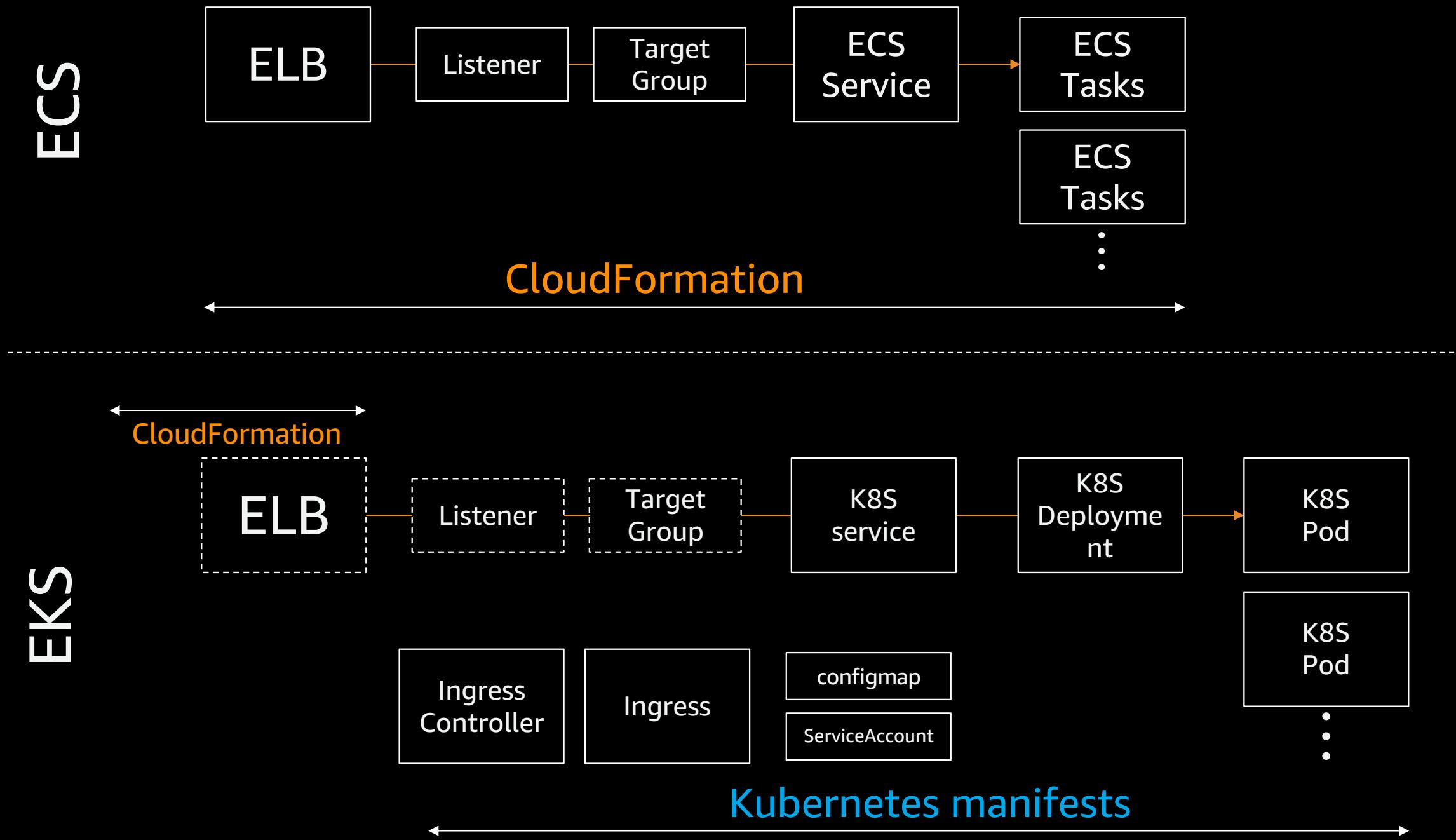
AWS Auto Scaling

- ECS Service Auto Scaling
- ECS Cluster Auto Scaling

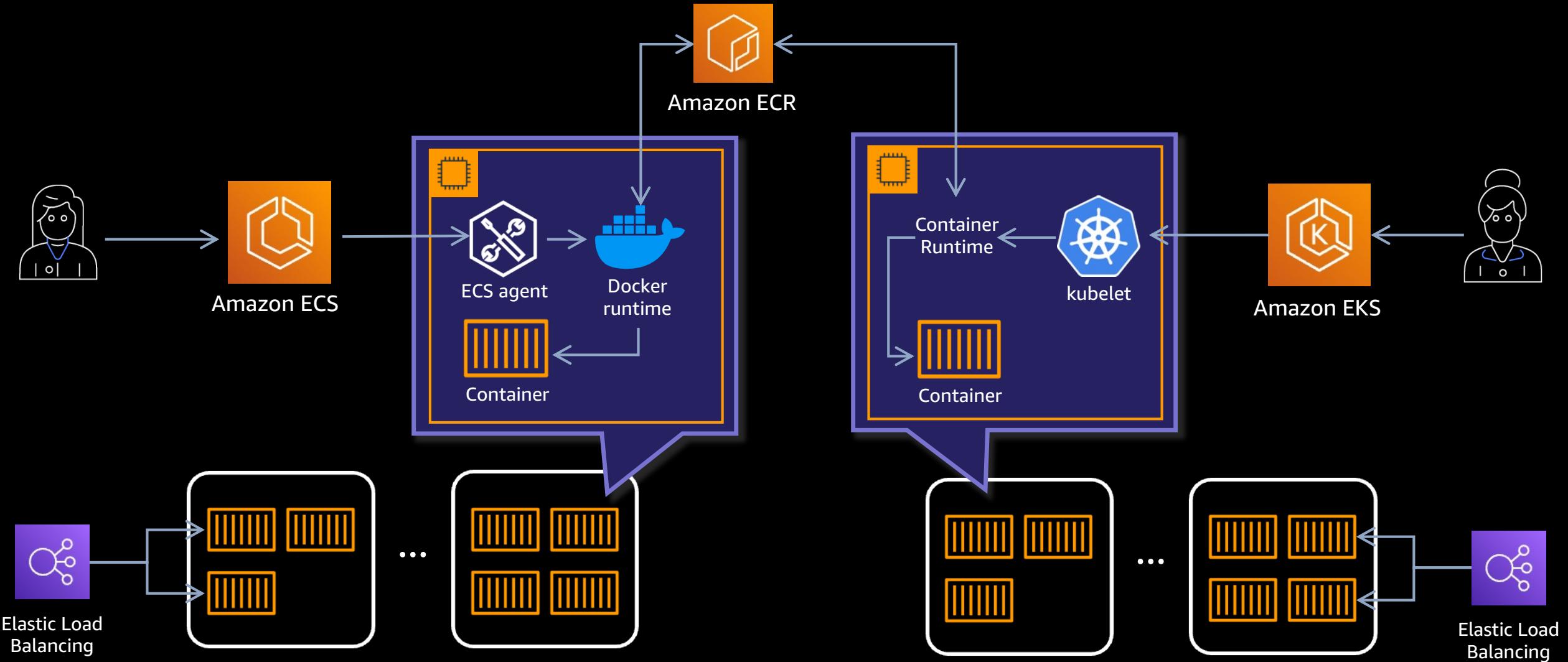
# 다른 서비스들과의 손쉬운 연동



# ECS vs EKS 컴포넌트



# AWS 관리형 컨테이너 서비스의 구조



# Lab 실습

## 실습환경 Link

<http://bit.ly/hansol-hol>

Event Hash : dad2-1df9ff5664-00



## Workshop Docs

<http://bit.ly/hansol-docs>



감사합니다