



Kubernetes Overview

Jooyoung Kim | Enterprise Solutions Architect

<https://www.linkedin.com/in/joozero/>



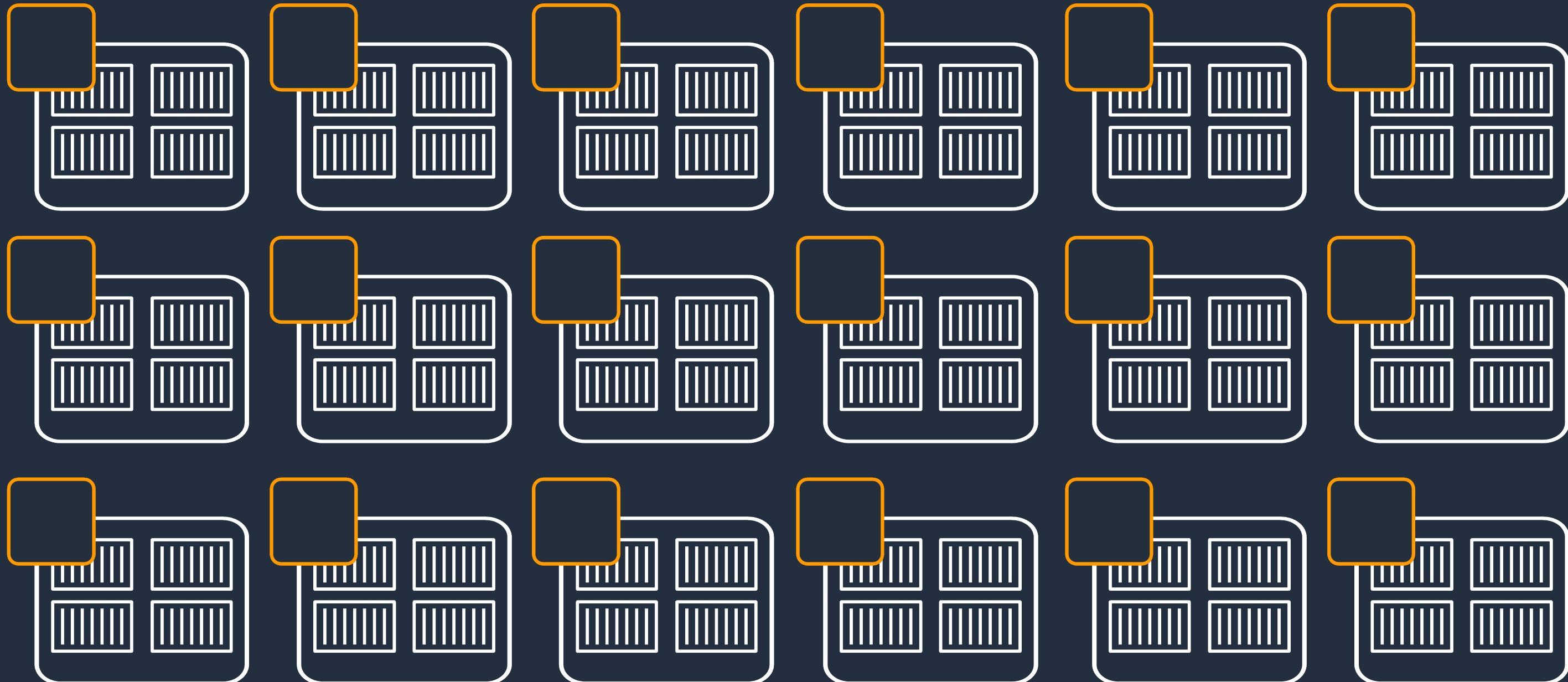
목차

- Kubernetes Architecture
- Kubernetes Object
- Additional Concept
- Best Practice
- Summary

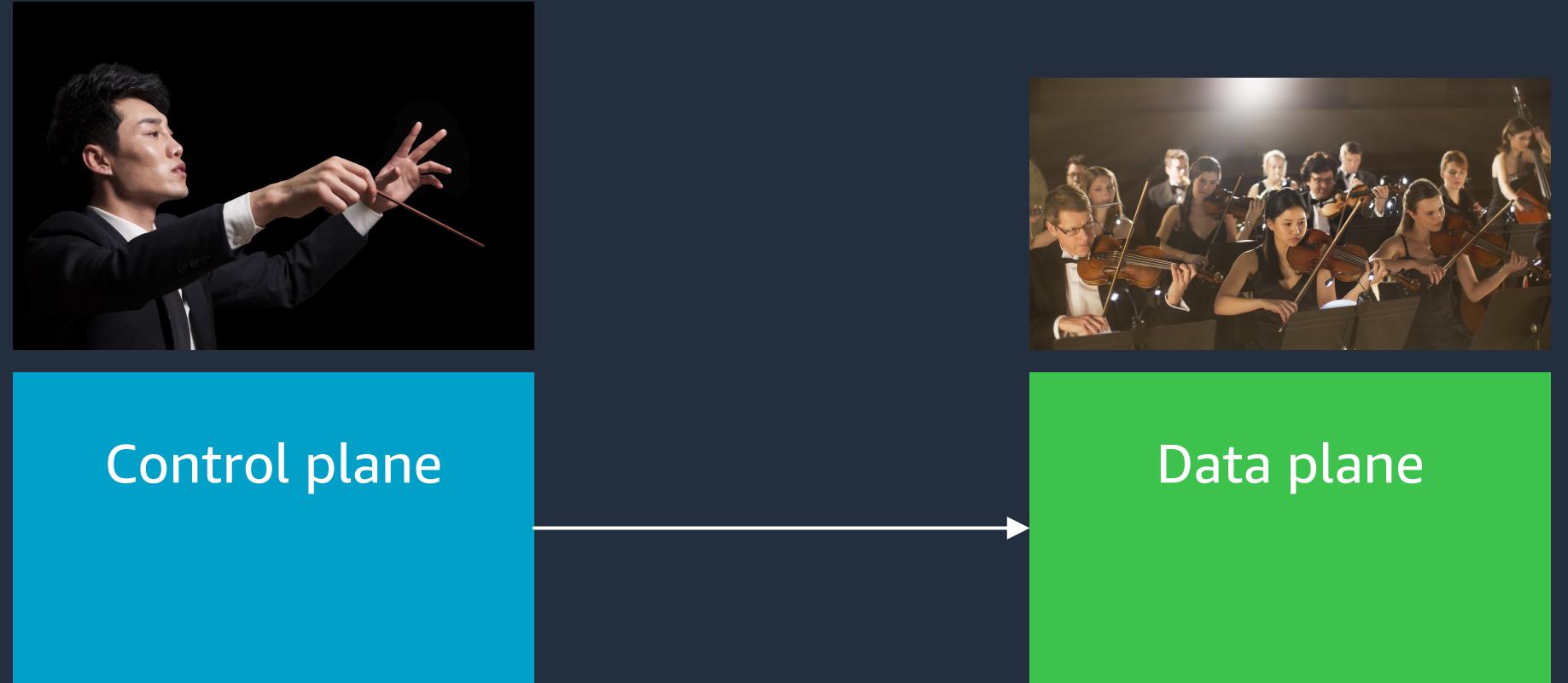
하나의 컨테이너로 시작



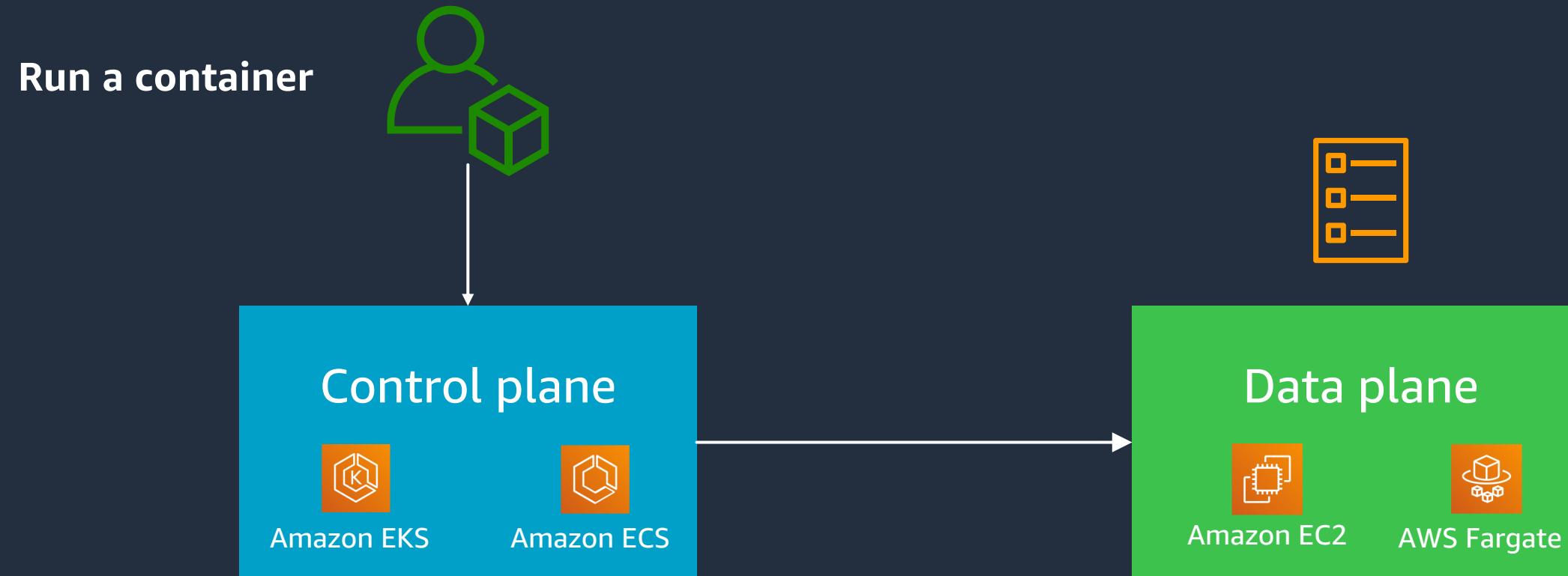
컨테이너가 아주 많아지게 된다면?



컨테이너 오케스트레이터

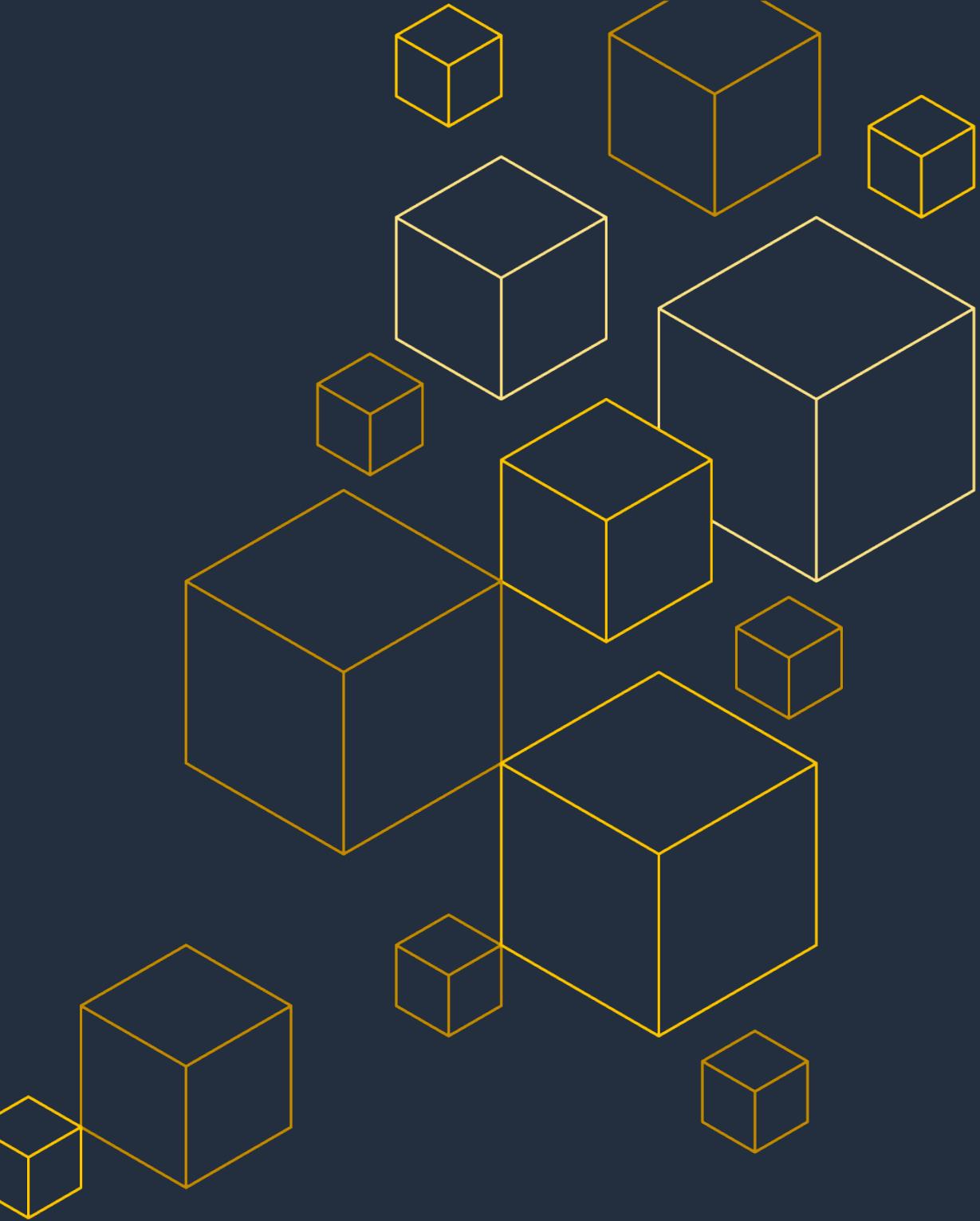


컨테이너 오케스트레이터



사용자가 원하는 상태(**Desired State**)로 동작하도록 관리(**Schedule**)하는 것

Kubernetes Architecture



Kubernetes

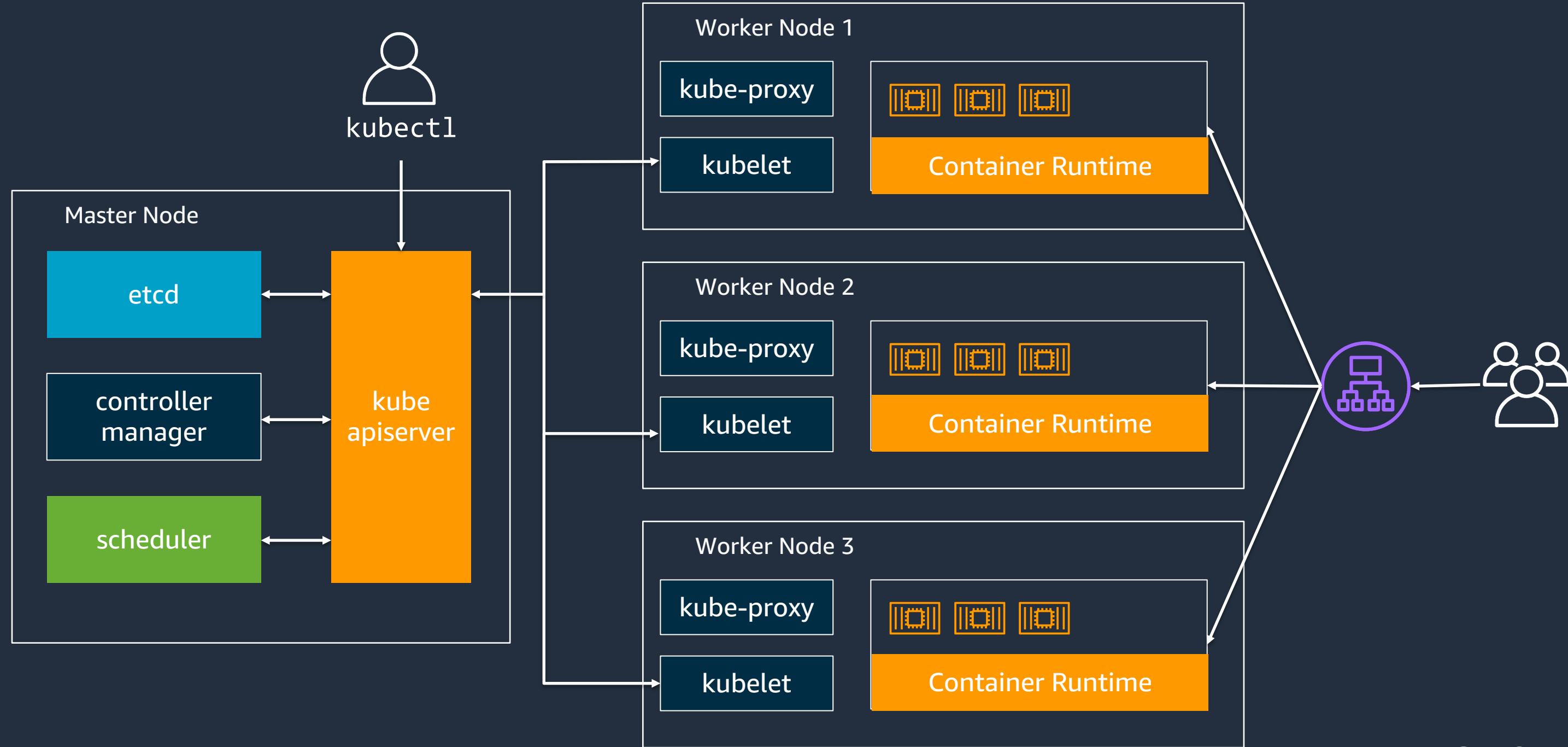


컨테이너화된 애플리케이션을 자동으로 배포, 스케일링 및 관리해주는 오픈 소스 시스템

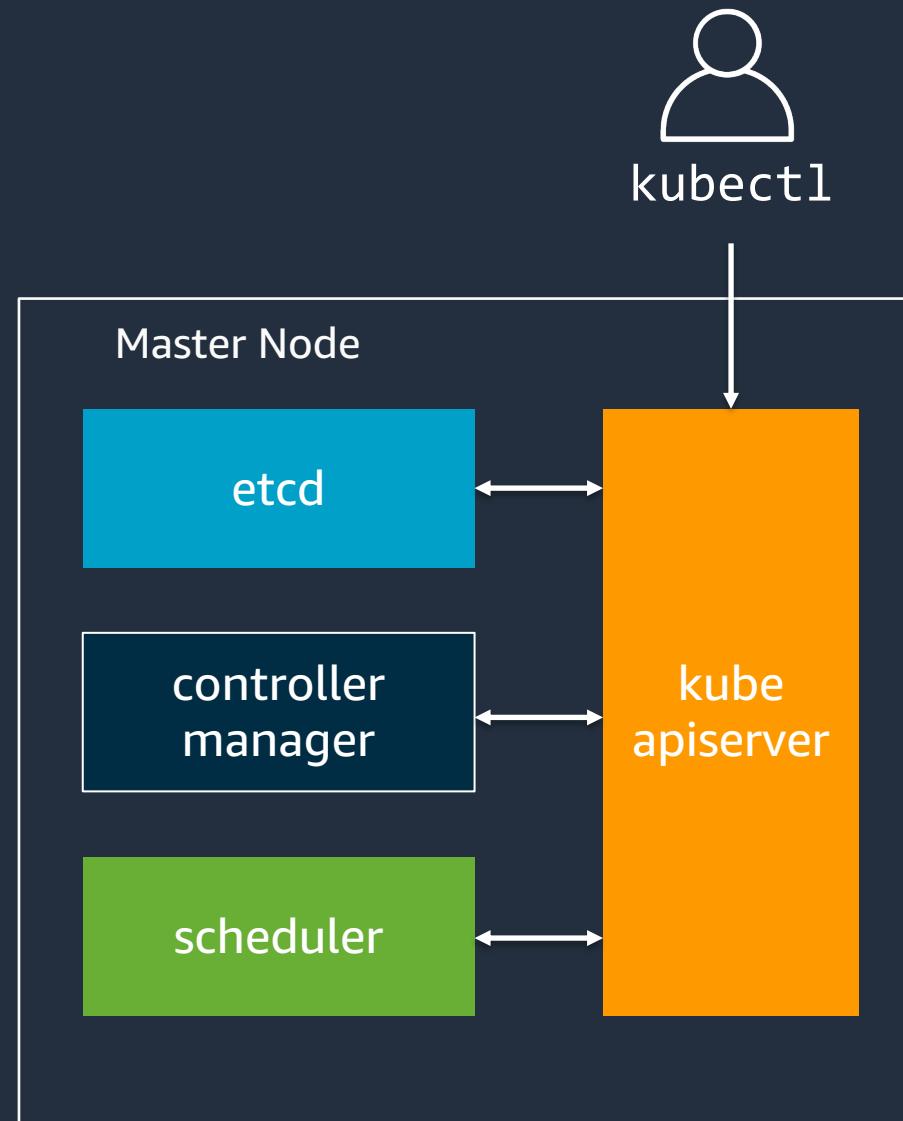
쿠버네티스가 제공하는 기능

- **서비스 디스커버리와 로드 밸런싱** : 컨테이너에 대한 트래픽이 증가할 때, 배포가 안정적으로 이뤄질 수 있도록 네트워크 트래픽을 로드 밸런싱
- **스토리지 오케스트레이션** : 원하는 스토리지 시스템을 자동으로 마운트할 수 있는 기능 제공
- **자동화된 롤아웃과 롤백** : 현재 상태를 원하는 상태로 설정한 속도에 따라 변경
- **자동화된 빈 배킹** : 각 컨테이너 필요한 CPU 및 메모리 양 설정
- **자동화된 복구** : 실패한 컨테이너를 다시 시작하고, 컨테이너를 교체하며 상태 검사에 응답하지 않는 컨테이너를 종료시킴
- **시크릿과 구성 관리** : OAuth 토큰 및 SSH 키와 같은 중요한 정보 저장 및 관리

Kubernetes 클러스터 아키텍처



컨트롤 플레인 구성 요소



- **apiserver**

- Kubernetes API를 노출하는 컴포넌트
- kubectl 등으로부터 리소스를 조작하라는 지시를 받음
- API 서버는 최종 사용자, 클러스터의 다른 부분 그리고 외부 컴포넌트가 서로 통신할 수 있도록 HTTP API를 제공

- **etcd**

- 고가용성을 갖춘 분산 키-값 스토어
- kubernetes cluster에서 발생하는 모든 이벤트를 저장

- **scheduler**

- 노드를 모니터링하고 애플리케이션(pod)을 배치할 적절한 노드를 선택

- **kube-controller-manager**

- Pod를 복제하거나 노드 운영 등 각 리소스를 제어하는 컨트롤러들을 감독하고 실행
- 노드 컨트롤러, 레플리케이션 컨트롤러, 디플로이먼트 컨트롤러 등

워커 노드 구성 요소

- **kubelet**

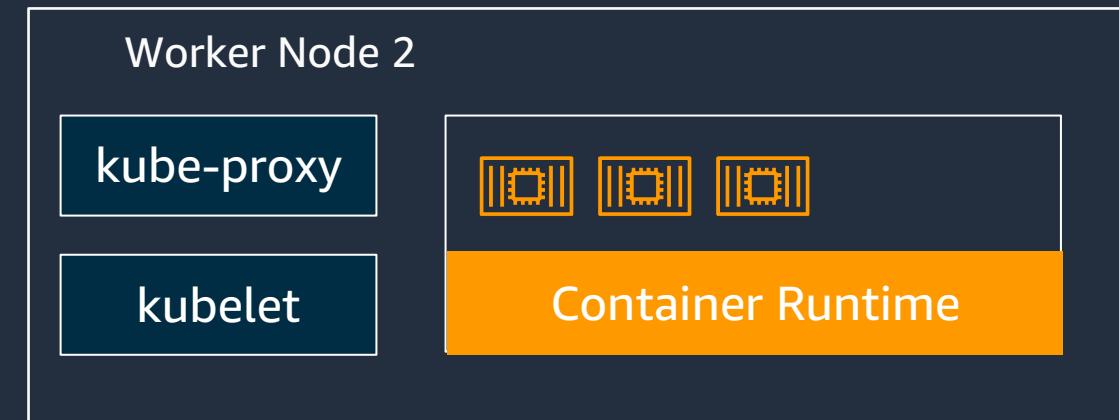
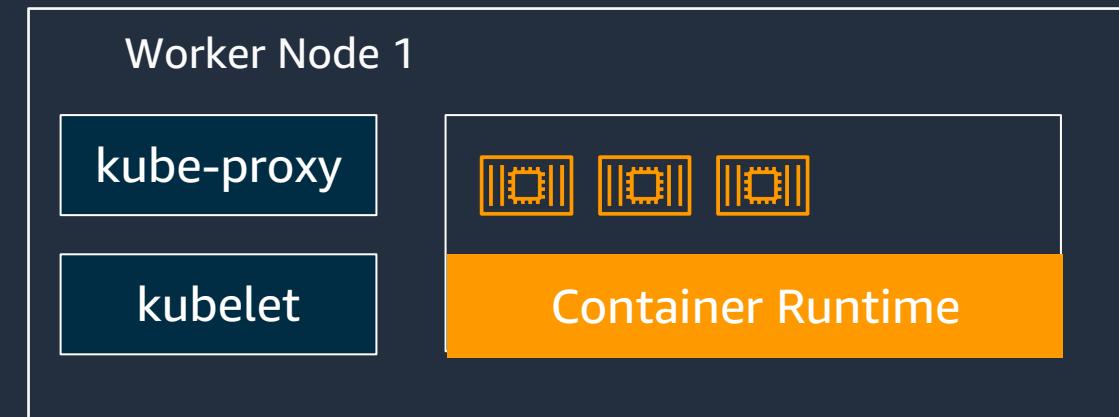
- 클러스터의 각 노드에서 실행되는 에이전트
- 컨트롤 플레인으로부터의 요청을 처리

- **kube-proxy**

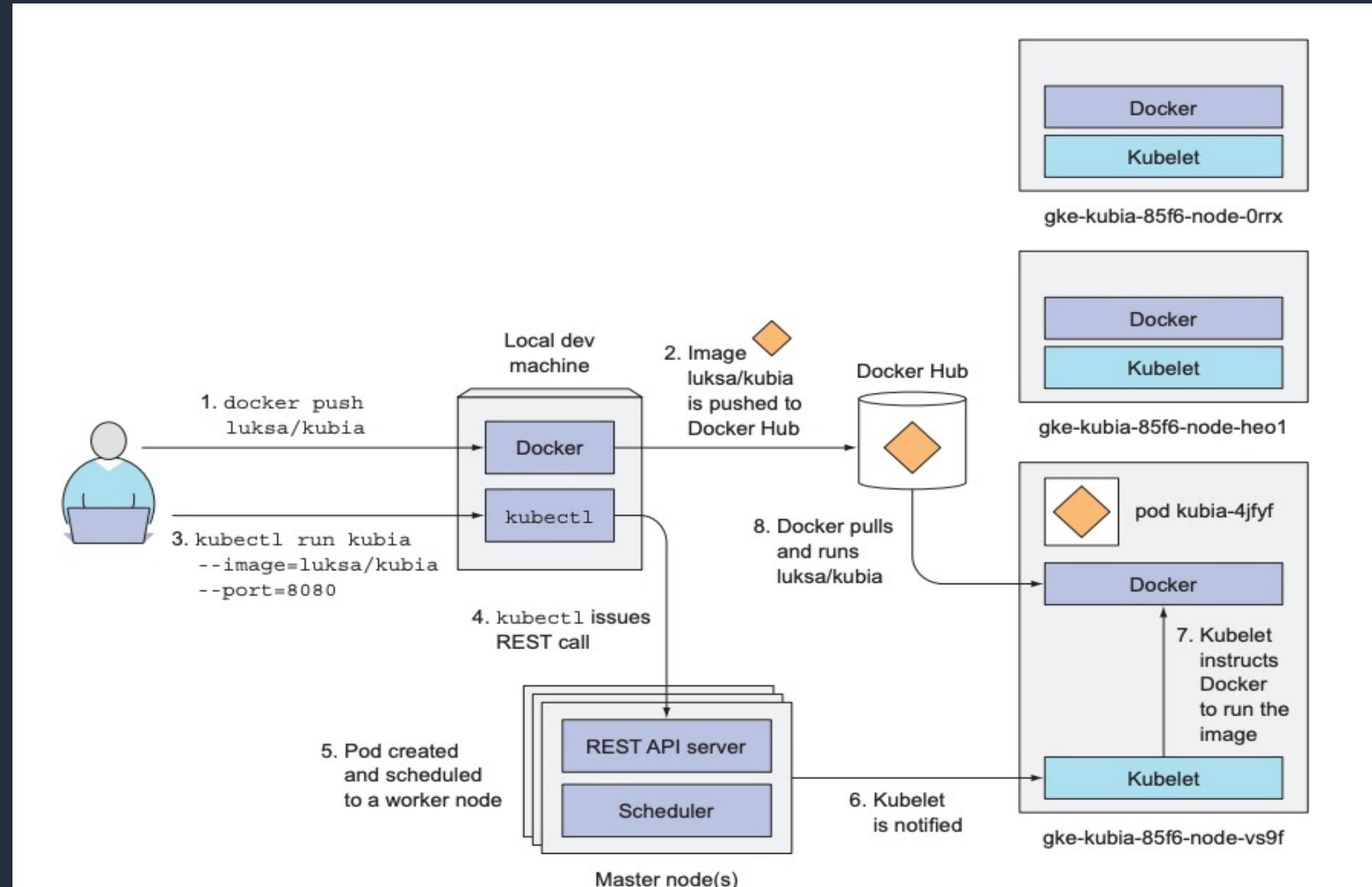
- 클러스터의 각 노드에서 실행되는 네트워크 프록시
- 노드의 네트워크 규칙을 유지 관리하여 내부 네트워크 세션이나 클러스터 바깥에서 파드로 네트워크 통신을 할 수 있도록 함

- **container runtime**

- 컨테이너 런타임은 컨테이너 실행을 담당하는 소프트웨어
- 도커(Docker), containerd 등

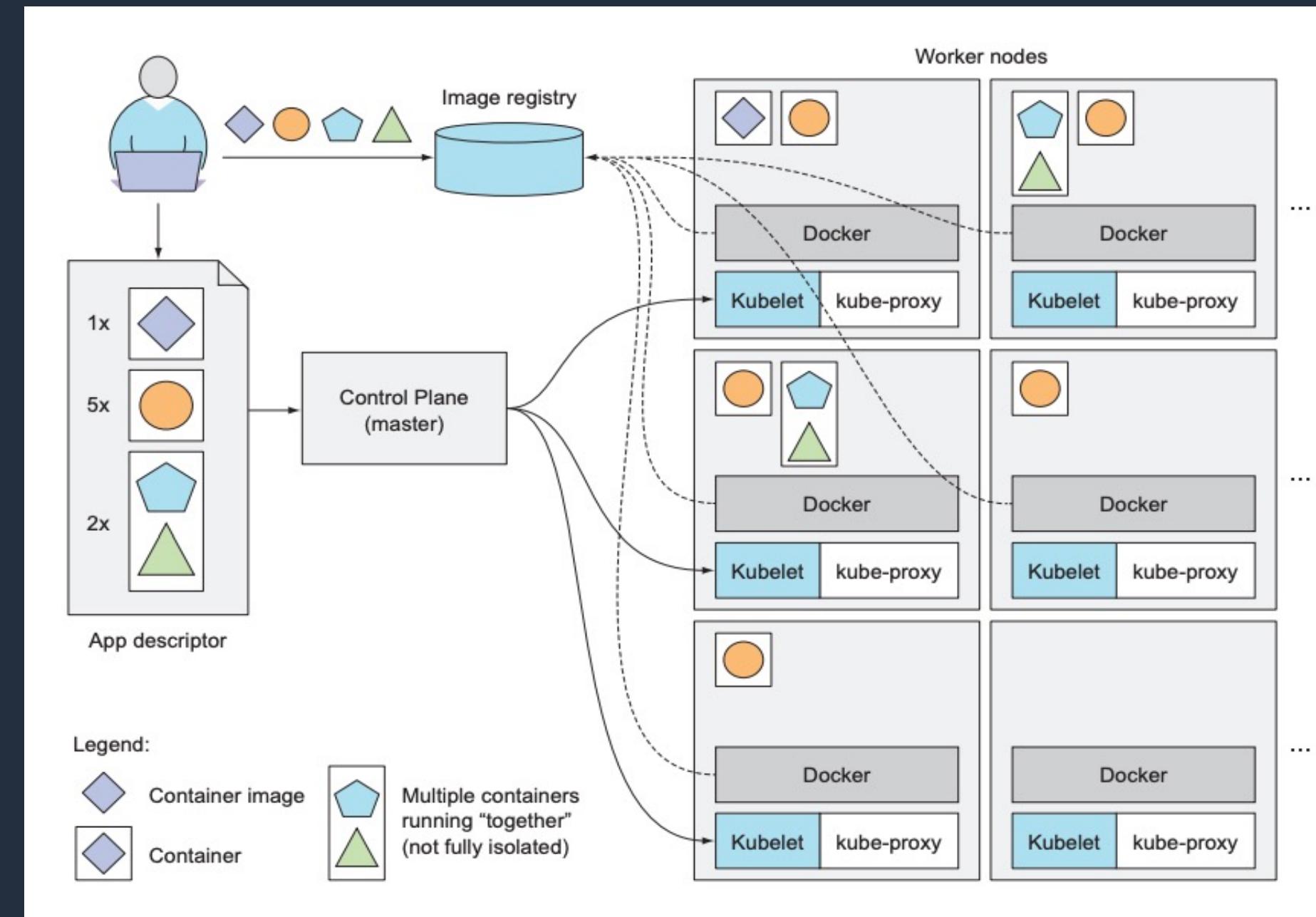


쿠버네티스 동작 예시



* Source: Kubernetes in Action, Marko Luksa, MANNING

쿠버네티스 동작 예시



* Source: *Kubernetes in Action*, Marko Luksa, MANNING

Kubernetes Object



Kubernetes Object

의도를 담은 레코드(record of intent)

오브젝트를 생성하게 되면, 쿠버네티스 시스템은 그 오브젝트 생성을 보장하기 위해 지속적으로 작동

오브젝트 설정을 위한 두 가지 중첩된 필드

- **Spec** : 오브젝트가 가져야 할 요구되는 상태(desired status)와 특징을 서술하는 곳
- **Status** : 오브젝트의 실제 상태를 기술하고 쿠버네티스 시스템에 의해 업데이트 됨. 쿠버네티스 control plane은 오브젝트의 실제 상태를 사용자가 서술한 상태와 일치시키기 위해 능동적으로 관리

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-nodejs-backend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-nodejs-backend
  template:
    metadata:
      labels:
        app: demo-nodejs-backend
    spec:
      containers:
        - name: demo-nodejs-backend
          image: public.ecr.aws/y7c9e1d2/joozero-repo:latest
          imagePullPolicy: Always
        ports:
          - containerPort: 3000
```

Yaml 파일

- 필수 구성 요소

apiVersion, kind, metadata, spec

- Label

파드와 같은 오브젝트에 첨부된 key-value 값으로 오브젝트의 특성을 식별하는데 사용됨

- Selector

효율적으로 레이블을 이용해 오브젝트를 쉽게 식별할 수 있게 함

- Namespace

쿠버네티스 자원에 대한 영역을 제공하고, 권한과 정책을

클러스터의 하위 섹션에 적용

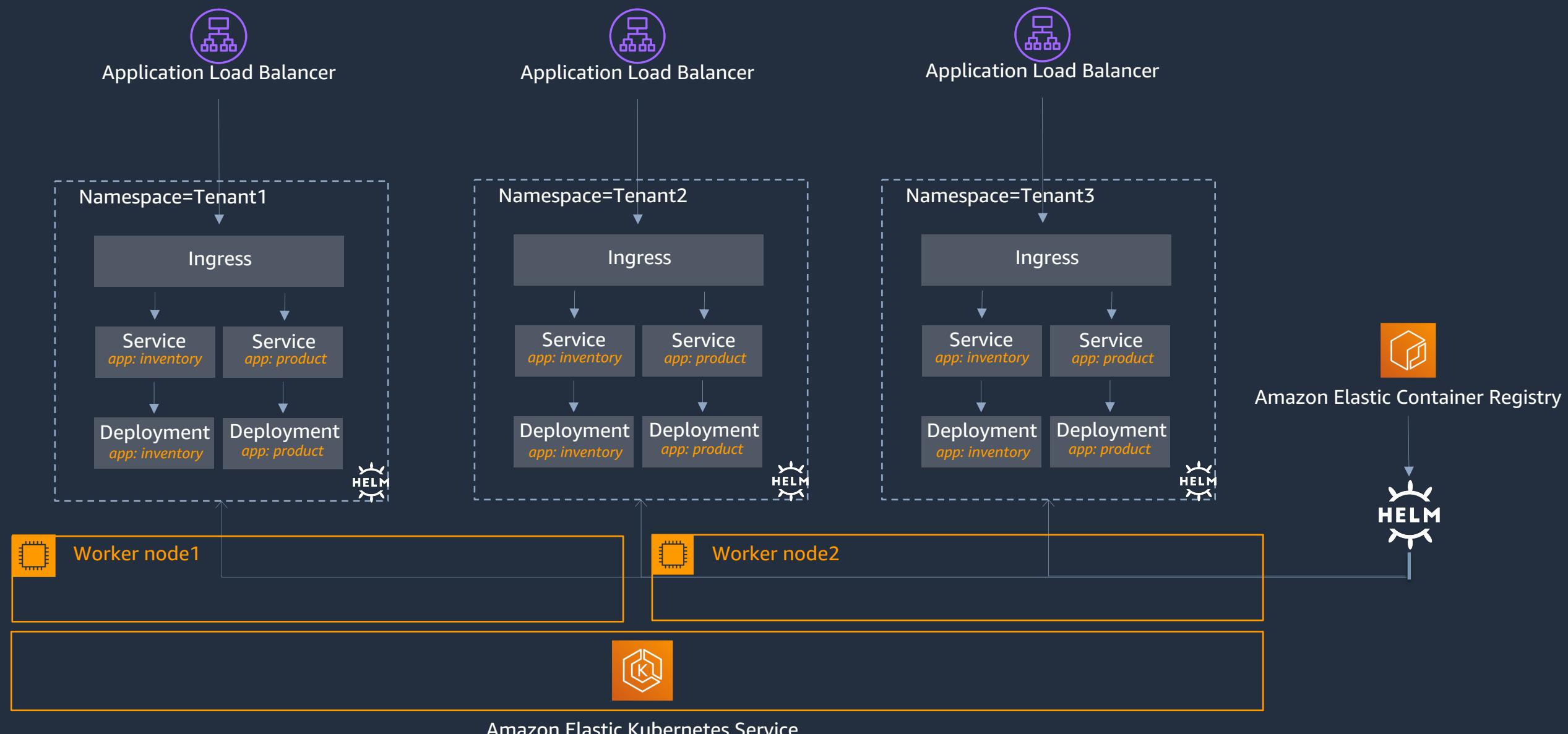
예: 개발 환경, 운영 환경

- Annotation

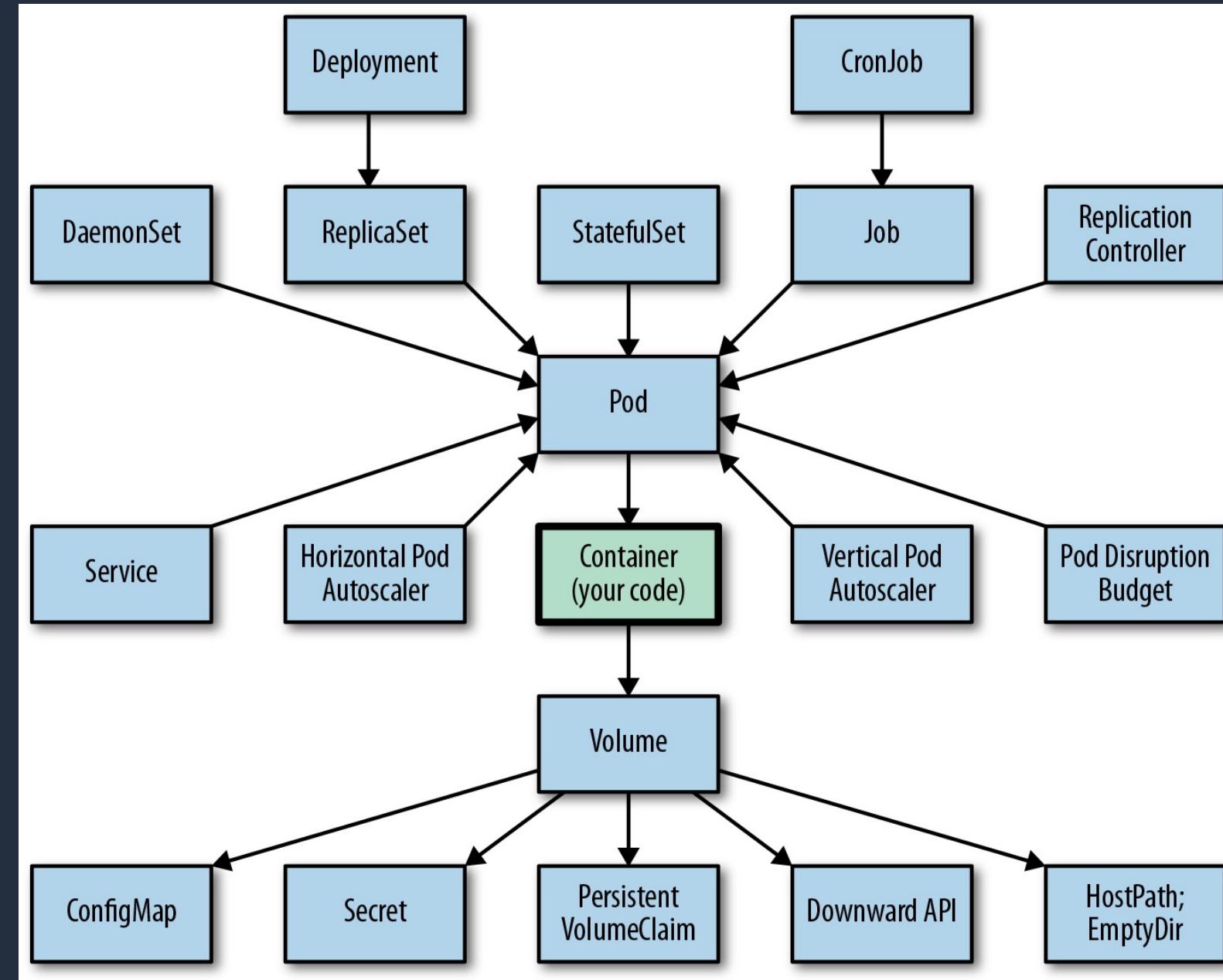
라벨처럼 map 형태로 구성되지만 사람보다는 machine을 위한 용도로 사용되며 검색이 불가능한 metadata를 지정하는데 사용

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-nodejs-backend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-nodejs-backend
  template:
    metadata:
      labels:
        app: demo-nodejs-backend
    spec:
      containers:
        - name: demo-nodejs-backend
          image: public.ecr.aws/y7c9e1d2/joozero-repo:latest
          imagePullPolicy: Always
        ports:
          - containerPort: 3000
```

Amazon EKS를 사용한 클러스터 아키텍처



k8s 사용자 입장에서 알아야하는 쿠버네티스 오브젝트들

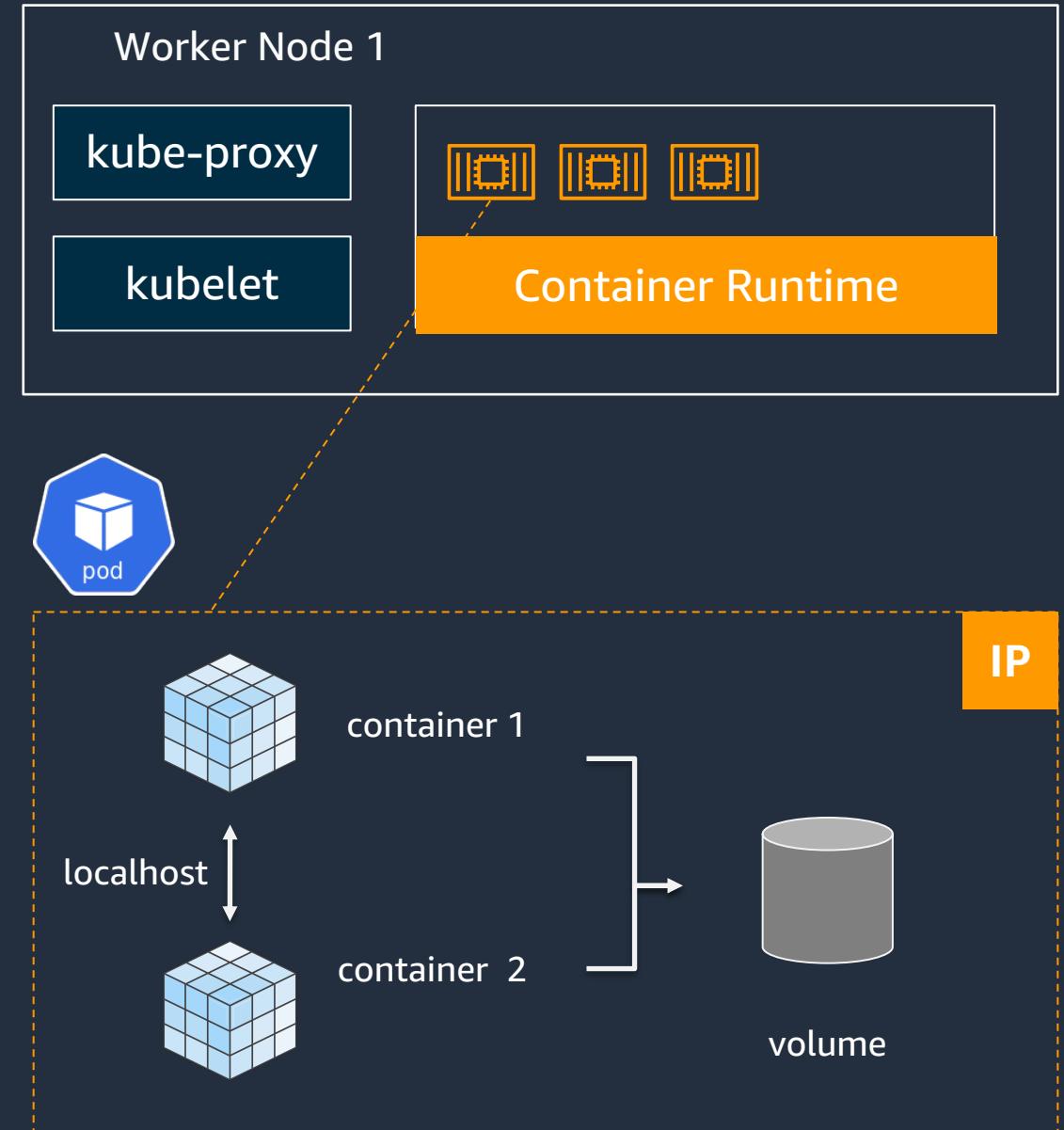


* Source: Kubernetes Patterns, Bilgin Ibryam & Roland Hub

Pod

쿠버네티스에서 생성하고 관리할 수 있는 배포 가능한
가장 작은 컴퓨팅 단위

- 애플리케이션의 최소 실행 단위
- 하나 이상의 컨테이너를 포함
애플리케이션 컨테이너(하나 또는 다수), 스토리지, 네트워크 등의 정보를 포함
즉, pod가 가지고 있는 컨테이너를 담은 가상 머신으로 볼 수 있음
- Pod의 특징
 - Pod에는 각각 고유한 private IP 할당
 - pod 안에 있는 컨테이너는 pod의 IP를 공유



Volume

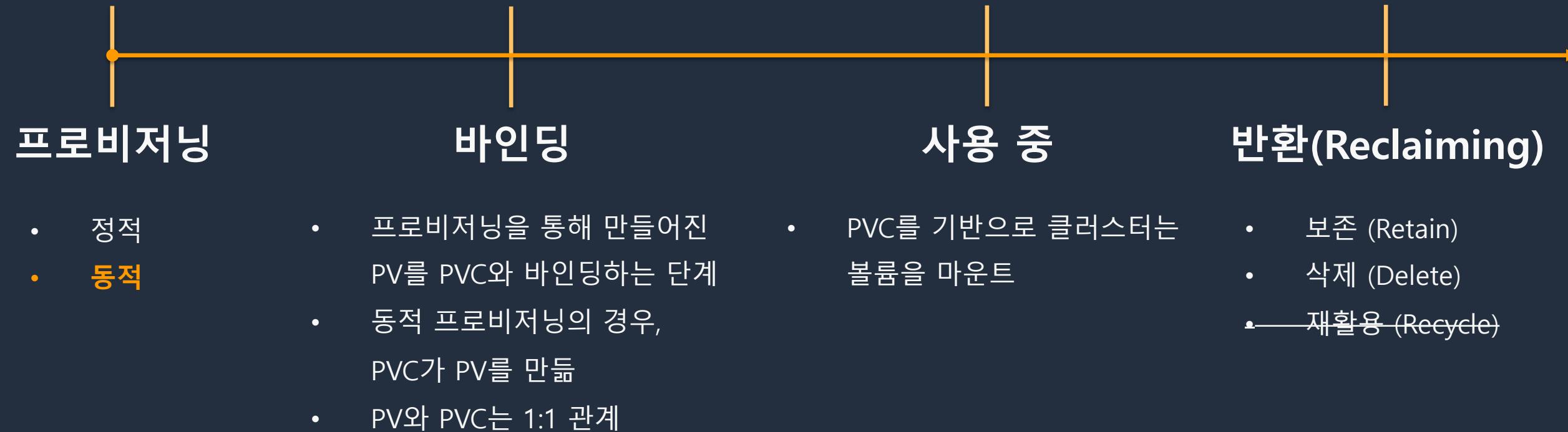
Volume Object의 필요성

- 컨테이너의 기본적인 상태 : 상태 없음(stateless)
따라서 컨테이너 내의 디스크에 있는 파일은 임시적이고, 이는 컨테이너가 재 시작할 때, 초기화 됨
- 하지만 일부 컨테이너는 재 시작하더라도 데이터를 유지할 필요가 있음

종류

- emptyDir** : 파드가 실행되는 호스트의 디스크를 임시로 컨테이너에 볼륨으로 할당(임시 디렉토리를 마운트)
- hostPath** : 파드가 실행된 호스트 파일이나 디렉토리를 파드에 마운트
- Persistent Volume** : 파드와 독립적으로 존재하는 클러스터 리소스. 관리자가 프로비저닝하거나 스토리지 클래스를 사용하여 동적으로 프로비저닝한 클러스터의 스토리지

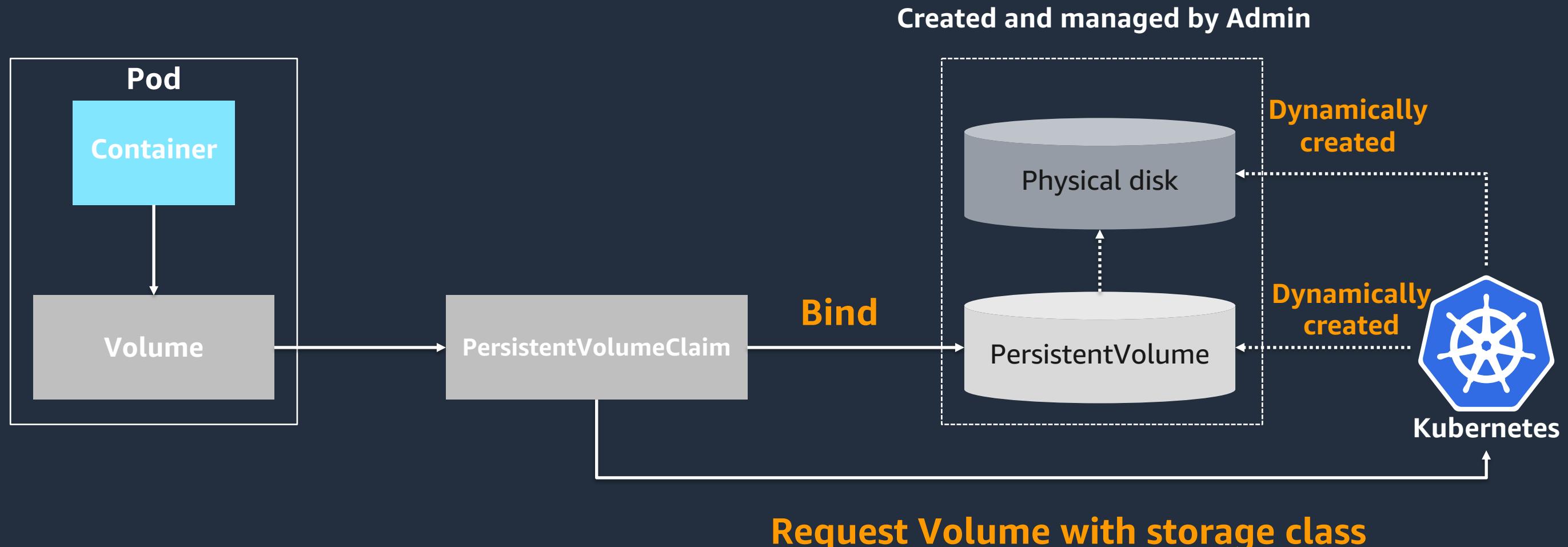
Persistent Volume Lifecycle



* PV(Persistent Volume)

* PVC(Persistent Volume Claim)

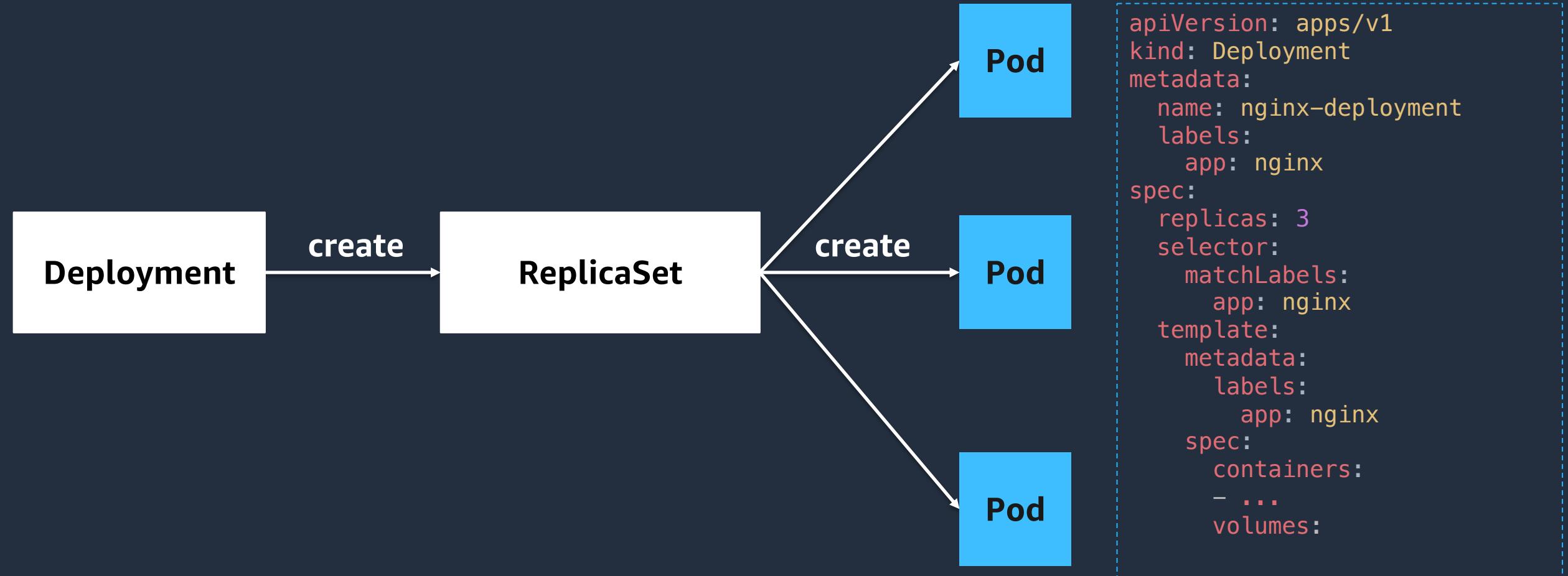
Persistent Volume Dynamic Provisioning



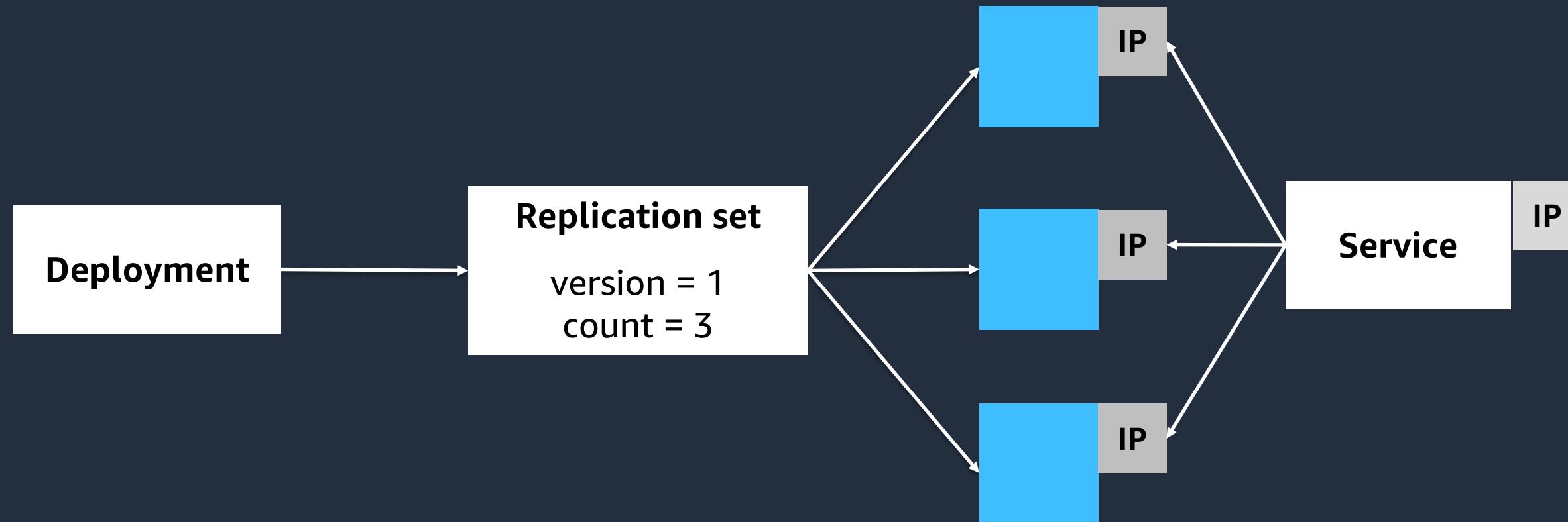
Deployment

애플리케이션 배포의 기본 단위 리소스

Deployment = ReplicaSet(의도한 파드 개수) + Pod(애플리케이션의 최소 실행 단위)

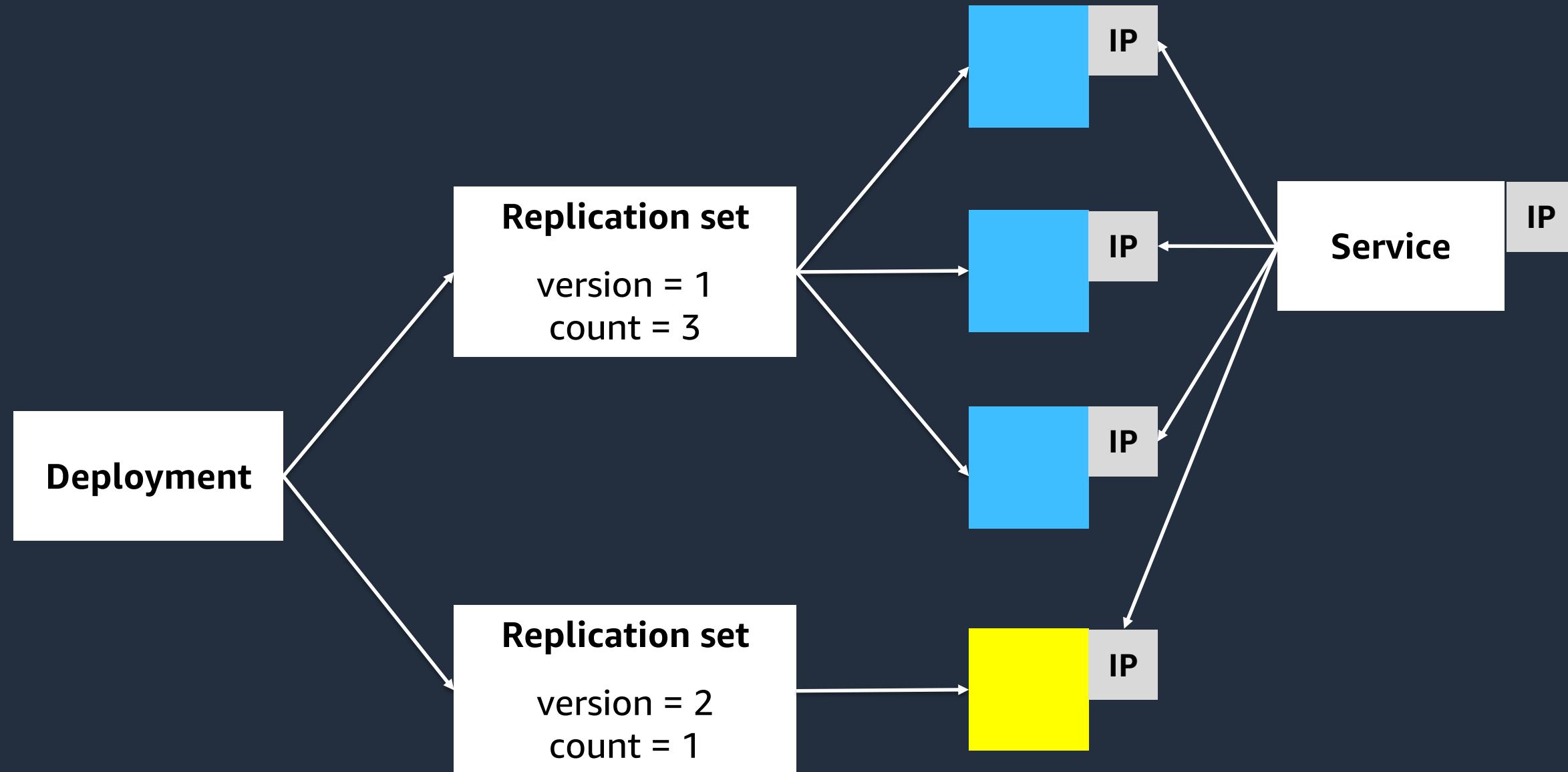


Deployment 업데이트 흐름

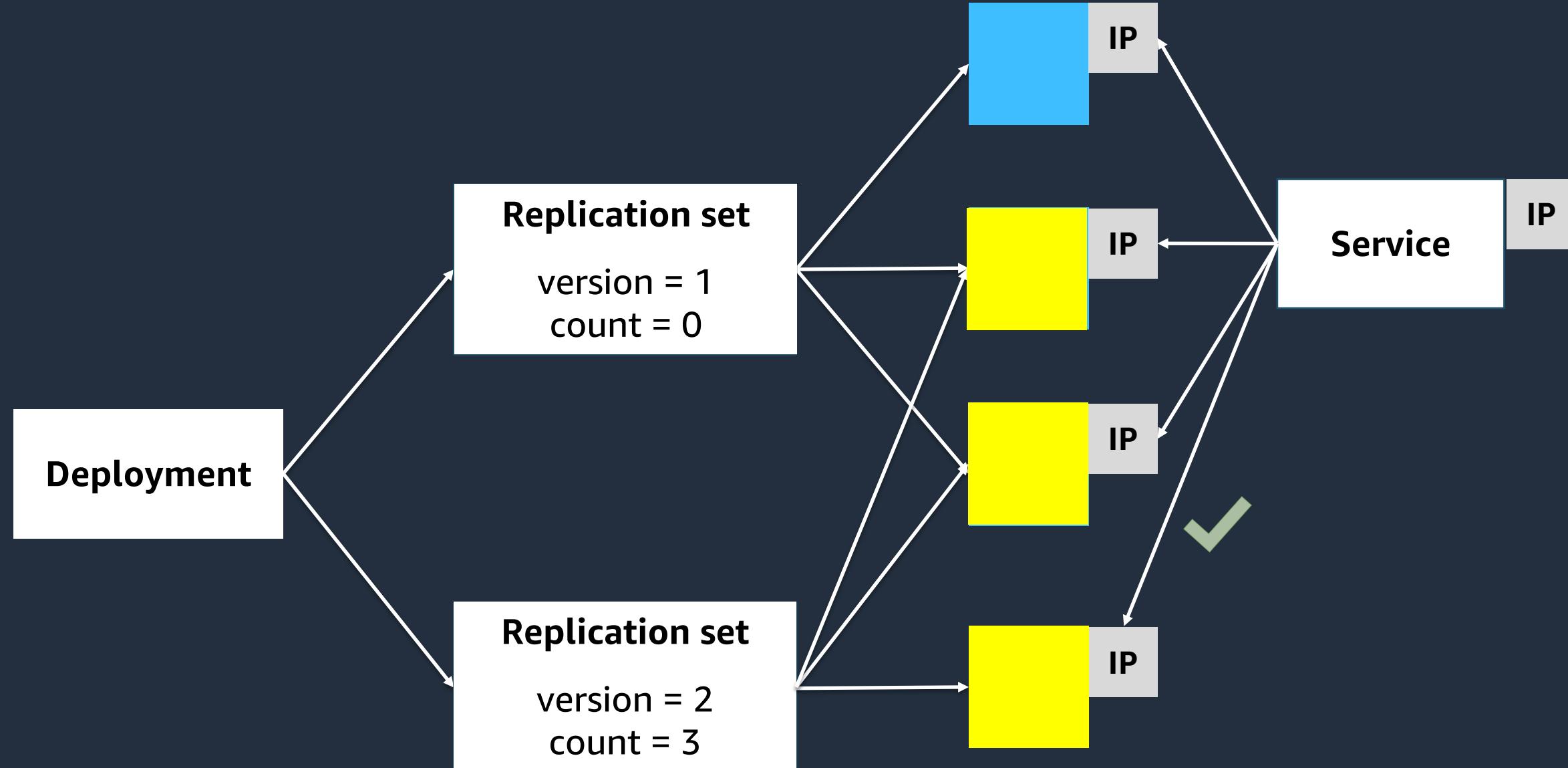


* Service는 Deployment와 함께 새로운 파드를 추가하거나 새로운 변경 사항을 관리합니다

Deployment 업데이트 흐름



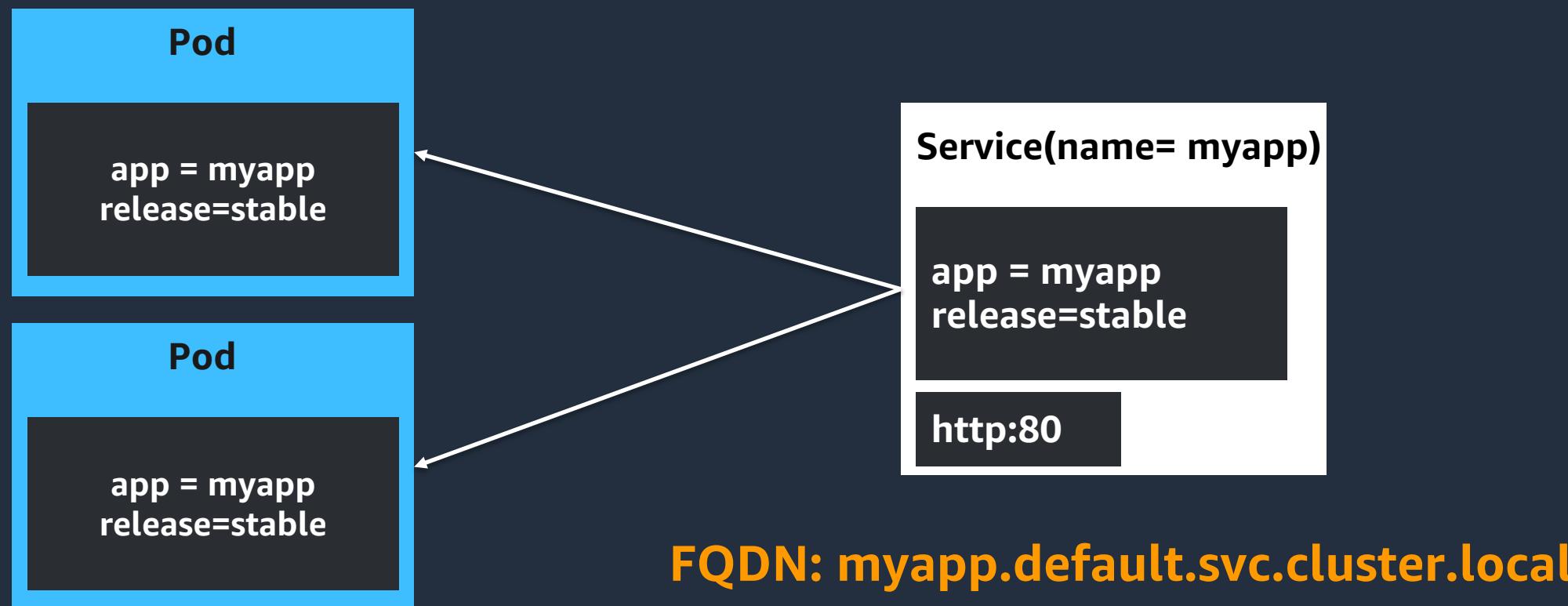
Deployment 업데이트 흐름



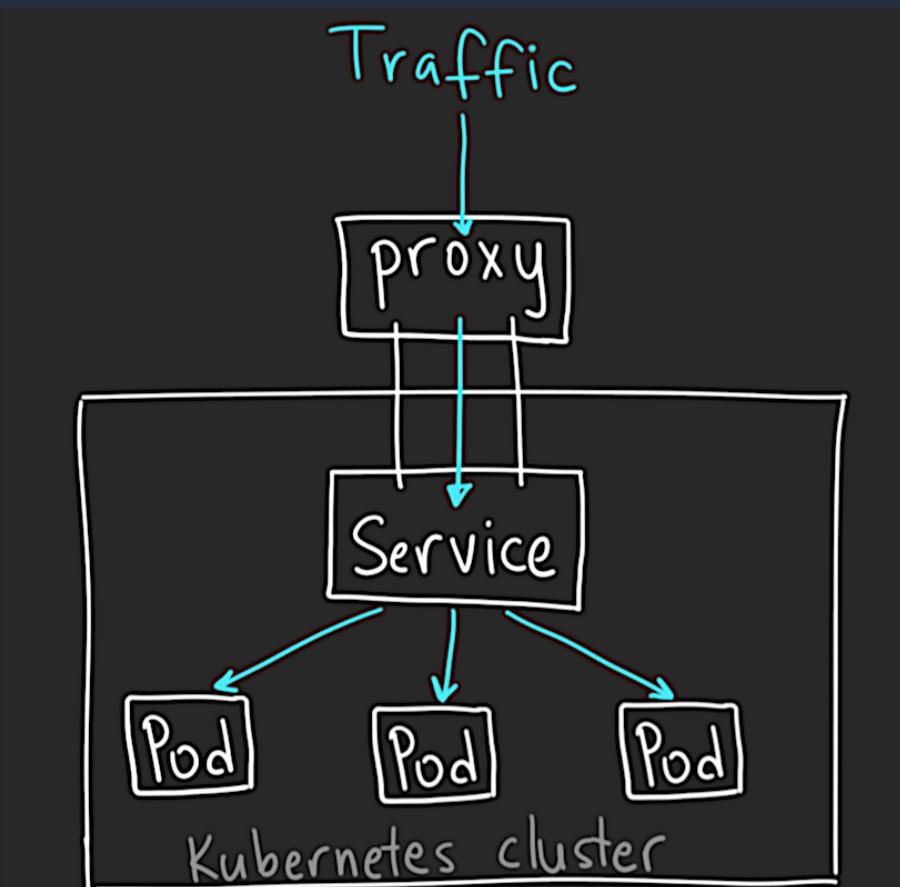
Service

Service Object의 필요성

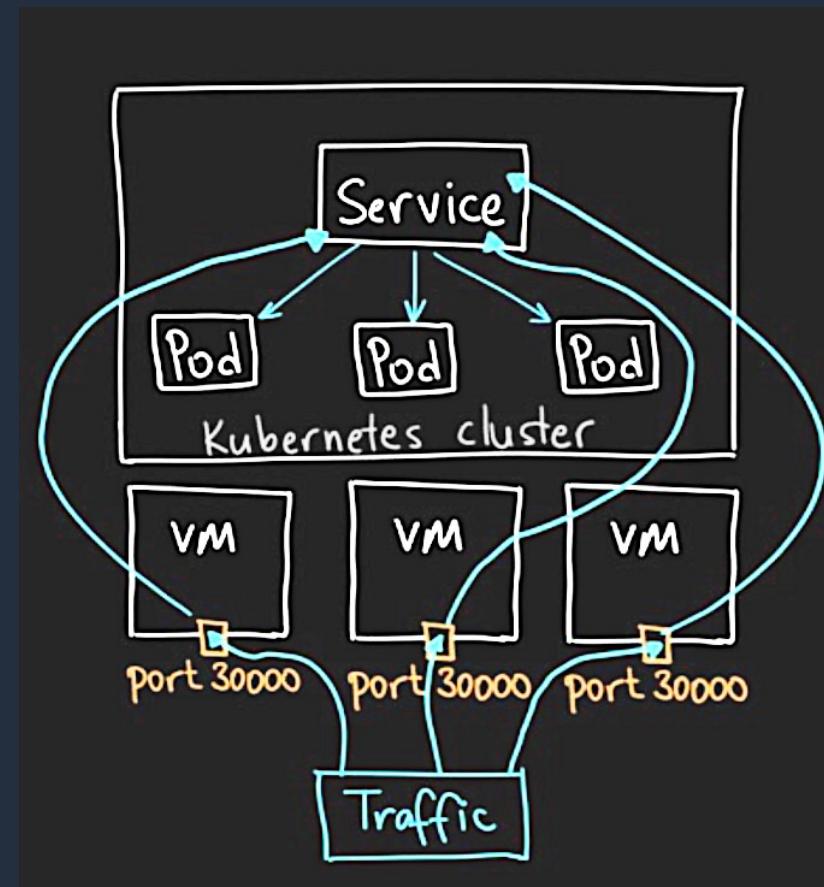
- 파드는 수명 주기가 짧은 일시적인 자원
이러한 상황에서 파드를 외부(인터넷과 같은) 네트워크로 노출하려면 수명 주기에 따라 변경되는 파드의 IP 주소를 찾고 파드가 구동하는 애플리케이션의 port 번호를 찾아 연결해야 함
- 요청을 파드에 라우팅하는 영구적인 IP 주소와 DNS 주소가 필요



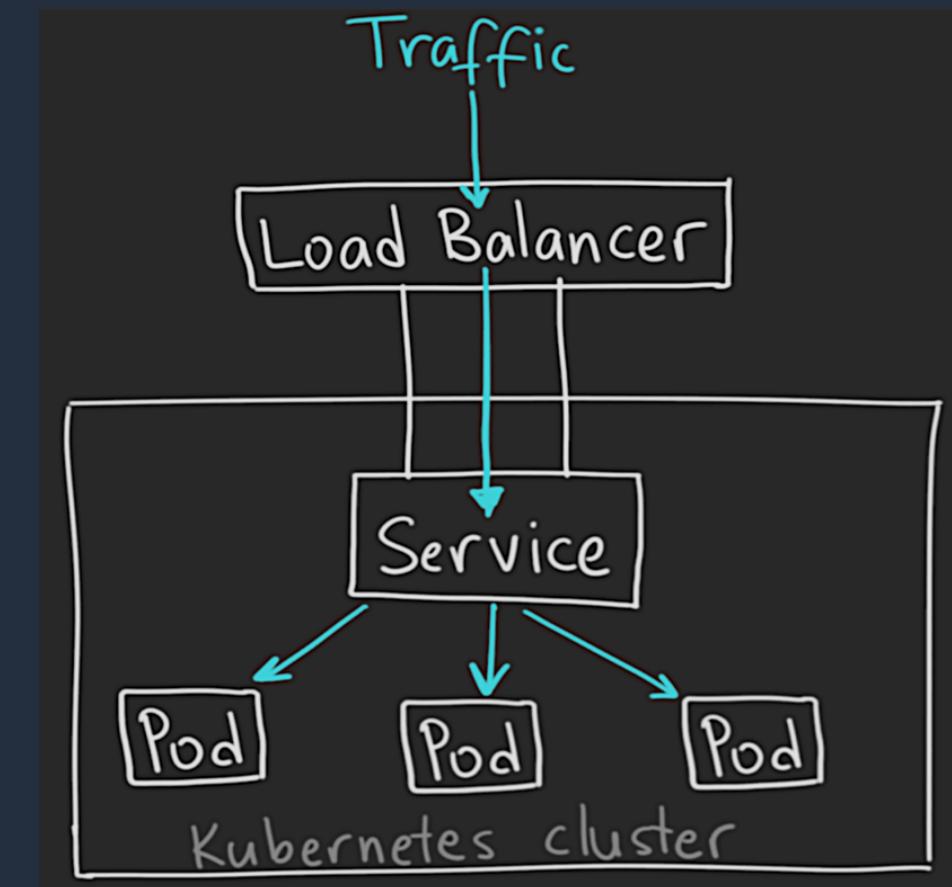
Kubernetes의 다양한 Service Object 종류



ClusterIP



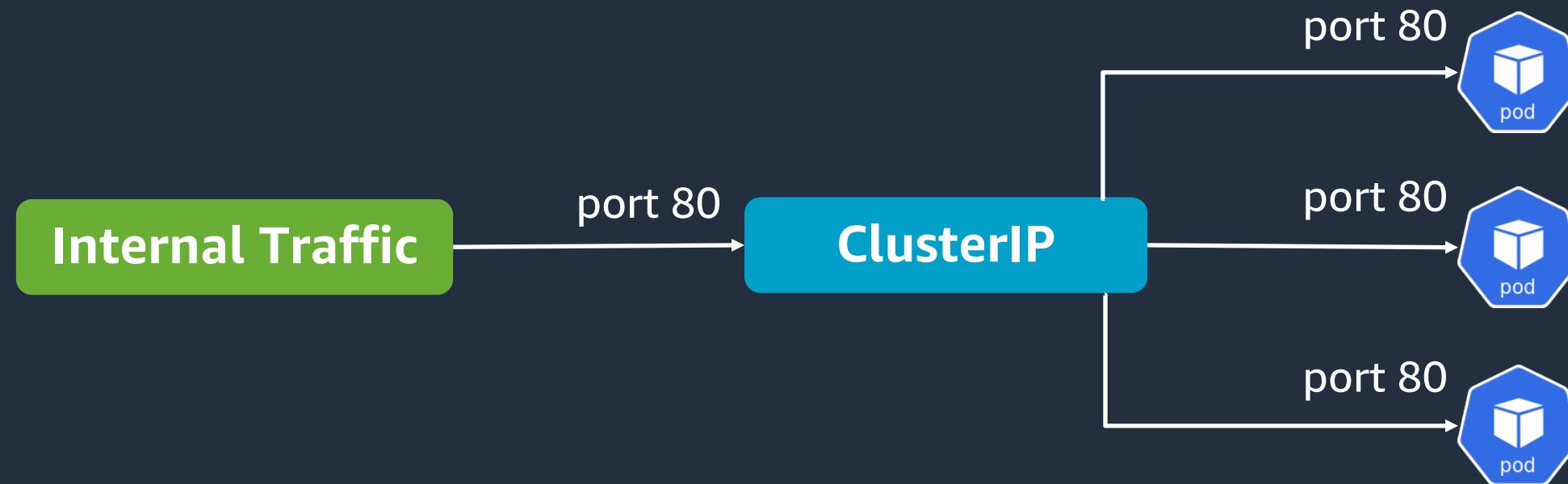
NodePort



LoadBalancer

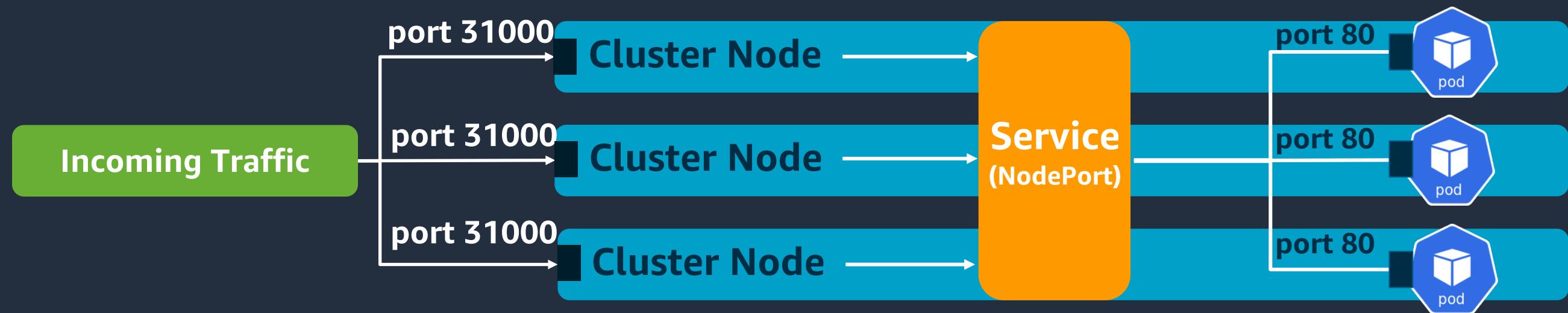
Service : ClusterIP

- Cluster-internal IP를 통해 Service를 노출
- 오직 클러스터 내부에서만 접근 가능
- kube-proxy를 통해 접근 가능
- 서비스를 디버깅하거나, 개발자 로컬에서 접근 또는 내부 dashboard를 보여줄 때 사용



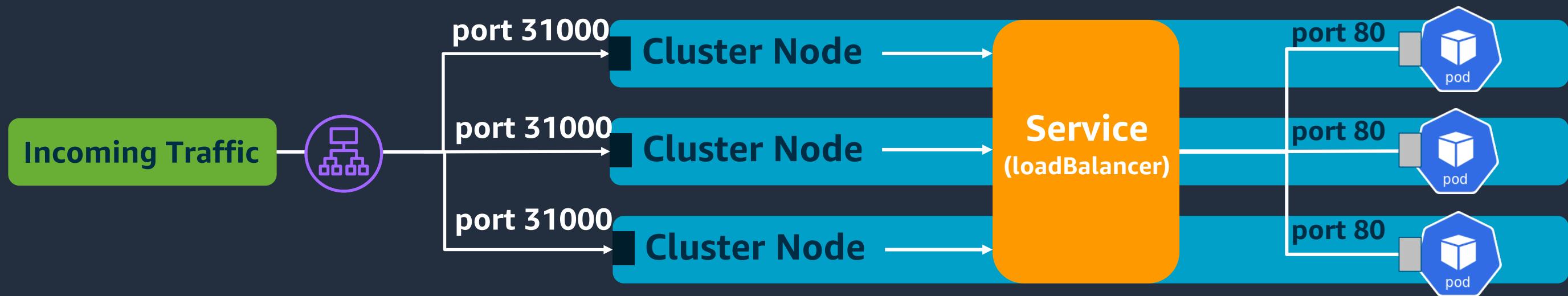
외부로 서비스 노출시키기 : NodePort

- NAT를 이용하여 클러스터 내 노드의 고정된 port를 갖는 IP로 service를 노출
- 클러스터 외부에서 접근은 <NodeIP>:<NodePort>
- port 당 하나의 service
- port 사용 범위: 30000-32767



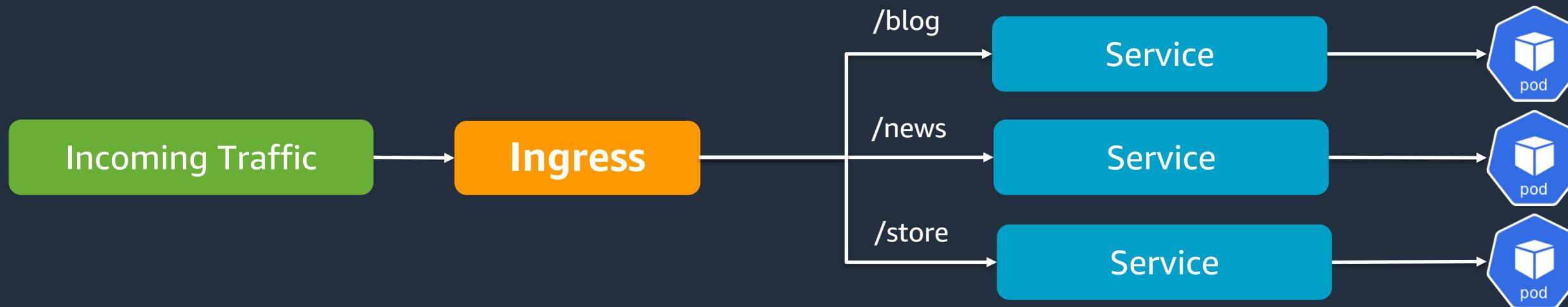
외부로 서비스 노출시키기 : LoadBalancer

- 클라우드 공급자의 로드밸런서를 이용해 service를 외부로 노출
- 클라우드에서 제공해주는 로드밸런서와 파드를 연결해서 그 로드밸런서의 IP를 이용해서 클러스터 외부에서 접근이 가능하게 만듦
즉, LB(ELB, NLB)로 노출된 각 service는 고유한 IP 주소를 가짐
- L4(TCP) or L7(HTTP) 레이어를 통해 service 노출



외부로 서비스 노출시키기 : Ingress

- Ingress는 Service 앞쪽에 위치하며 router 및 entry point 역할을 하는 별도의 k8s 자원
- 외부에서 접근 가능한 URL, Load Balancing, SSL Termination, 이름 기반 가상 호스팅 등을 통해 **Service**에 대한 **HTTP(s)** 기반 접근을 제공
- Ingress가 작동하려면 클러스터에 하나 이상의 실행 중인 **Ingress Controller**가 있어야 함
예: AWS Load Balancer Controller, NGINX Ingress
- 단일 외부 로드 밸런서를 통해, 여러 Service로 fan out 가능



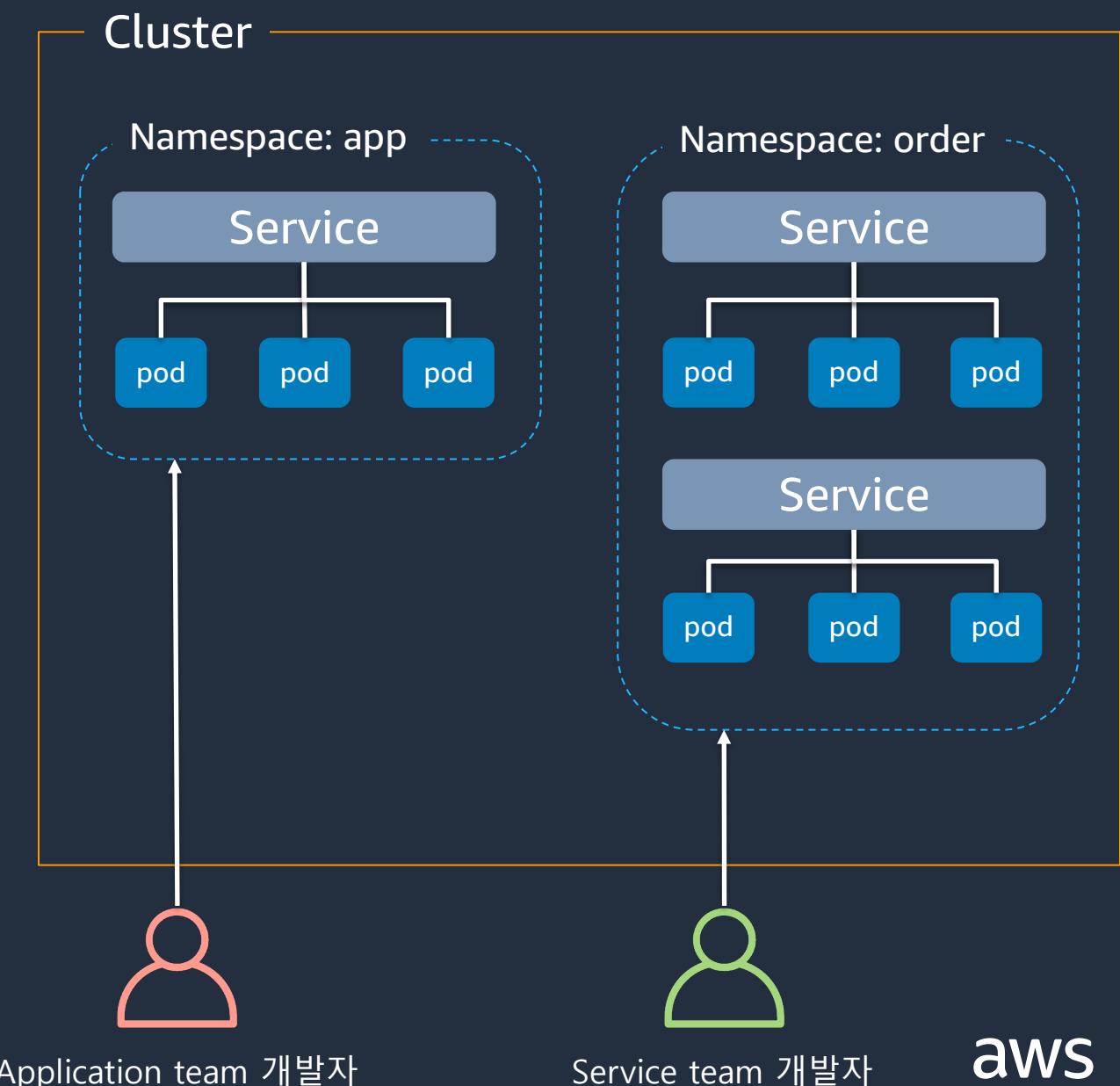
RBAC

RBAC(Role-based access control)의 필요성

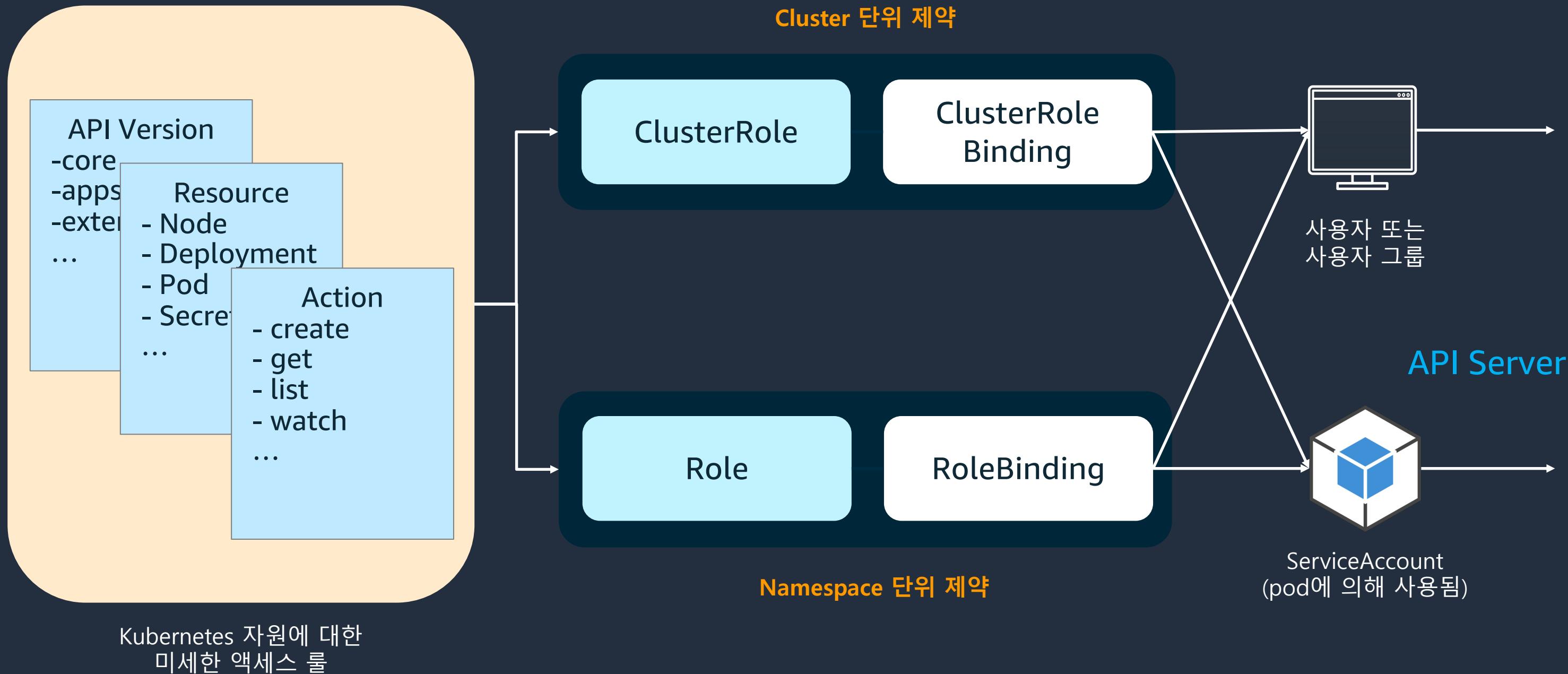
- 서비스의 규모가 커질 수록 예기치 않은 실수, 장애로부터 클러스터를 보호하기 위한 장치가 필요 이를 위한 가장 첫 번째 솔루션이 접근 제어
- 역할 기반 접근 제어(RBAC)**을 사용하여 특정 작업을 수행할 사용자 혹은 Service Account를 제어

종류

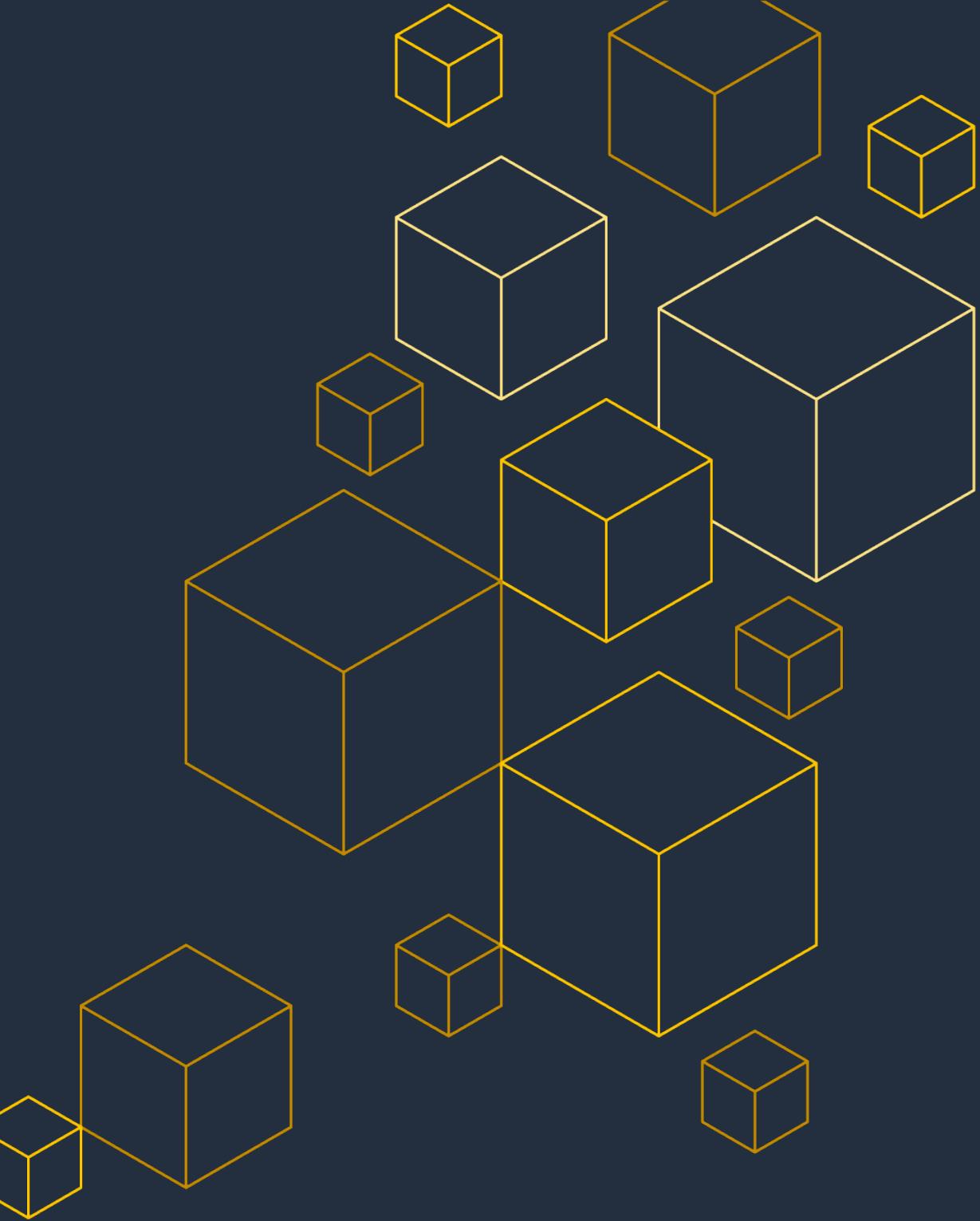
- RBAC Role, ClusterRole



RBAC



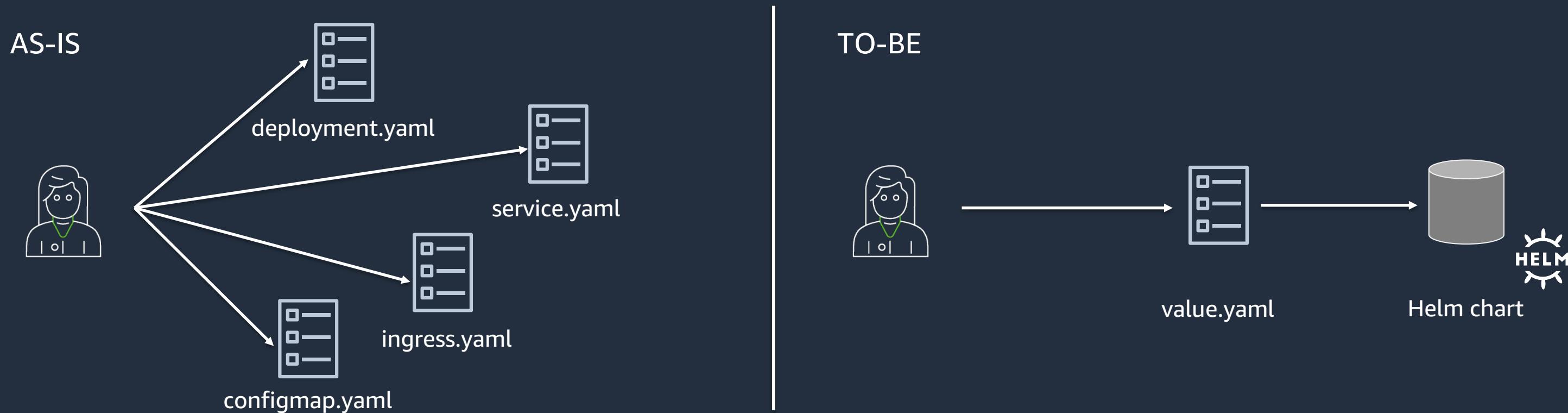
Additional Concept



Helm

쿠버네티스 용 패키지 매니저로 쿠버네티스 클러스터에 애플리케이션을 배포하는 도구

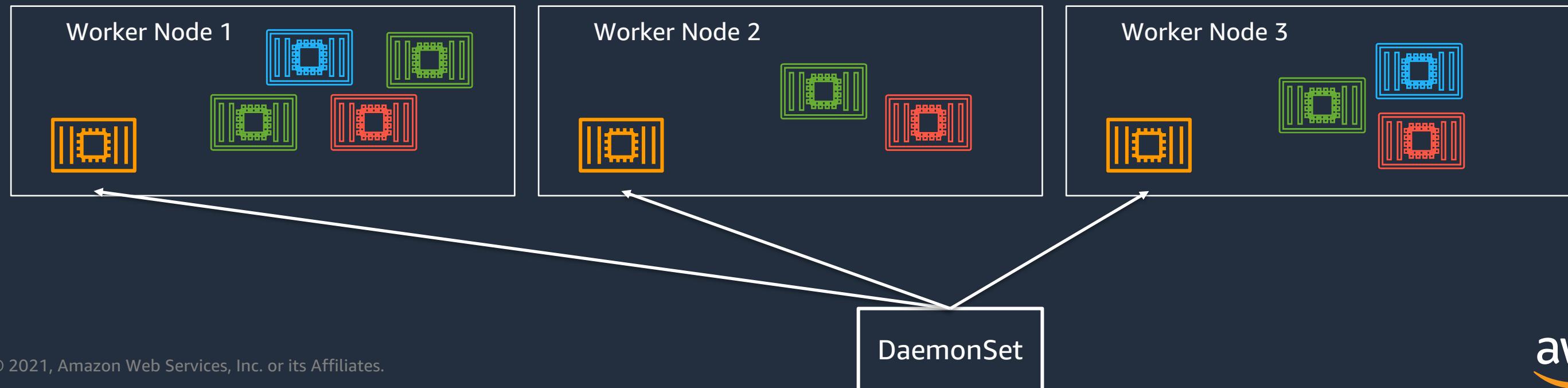
- Chart(yaml 파일들의 집합)라는 개념으로 애플리케이션을 정의하고 관리
- Chart들이 저장되어 있는 chart repository와 연계해서 쿠버네티스에 차트를 설치 및 삭제



DaemonSet

클러스터 전체에 파드를 띄울 때 사용하는 컨트롤러

- 데몬셋으로 파드를 실행하면 클러스터 내에 새로운 워커 노드가 추가 되었을 때, 자동으로 그 노드에 데몬셋으로 파드가 실행 됨
- 클러스터 전체에 항상 실행시켜 두어야 하는 파드
예: 로그 수집기, 모니터링 툴



kubectl 기본 명령어

```
# 상태 설정하기
```

```
$ kubectl apply
```

```
# 리소스 목록 보기
```

```
$ kubectl get pod # Pod 조회로 이름 검색
```

```
# 리소스 상세 보기
```

```
$ kubectl describe pod demo-nodejs-backend-65d857bbb5-7vk97
```

```
# 리소스 삭제
```

```
kubectl delete
```

```
# 컨테이너 로그 확인 & 명령어 전달
```

```
kubectl logs demo-frontend-559dd6dff5-xvg8k
```

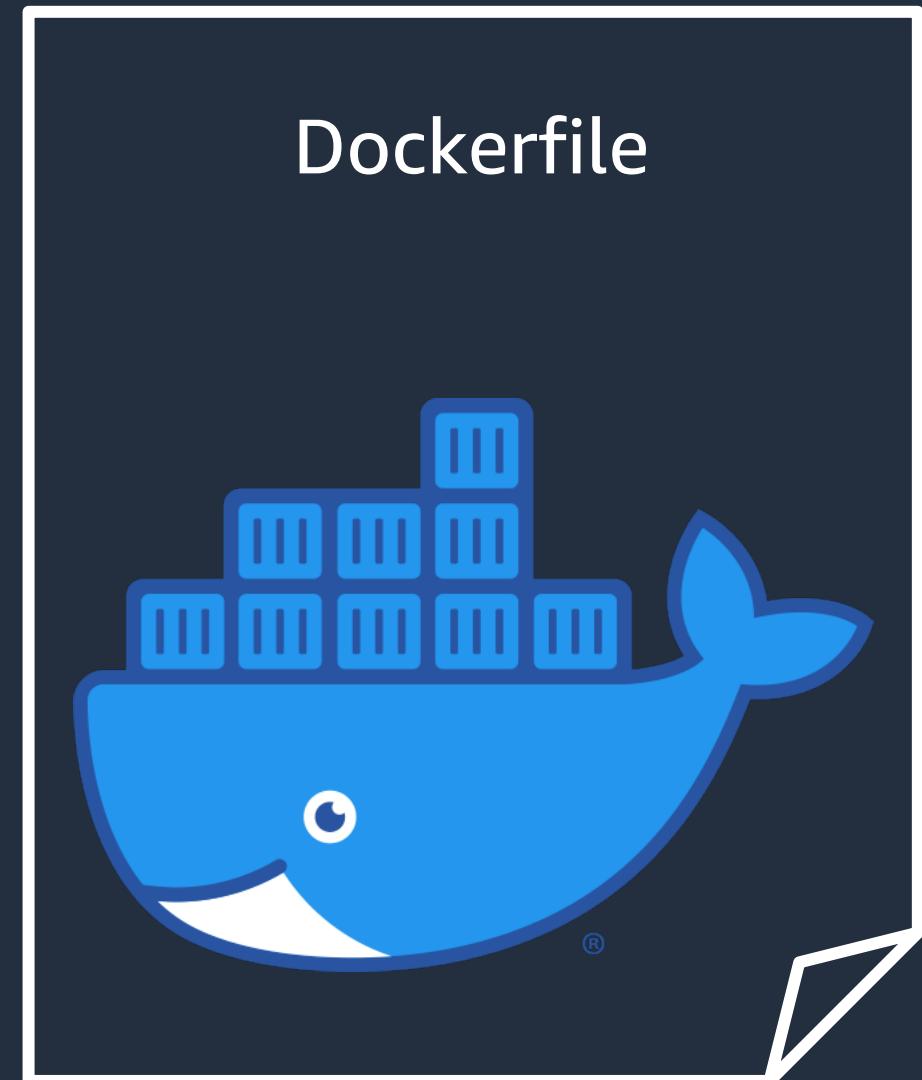
```
kubectl exec -it demo-nodejs-backend-65d857bbb5-n7lsg -- bash # 컨테이너 접속
```

K8S Best Practice



이미지 크기는 최대한 작게

- Alpine과 같이 가벼운 이미지 사용 권고
- 순서의 중요성
 - Docker build cache를 적극 활용 필요
 - Dockerfile에서 소스 코드는 최대한 늦게 추가
- 불필요한 패키지 및 파일 설치하지 않기
 - apt 패키지 매니저 사용시 –no-install-recommends 옵션
 - .dockerignore로 불필요한 파일 복사 막기
- Multi-stage build 사용할 것

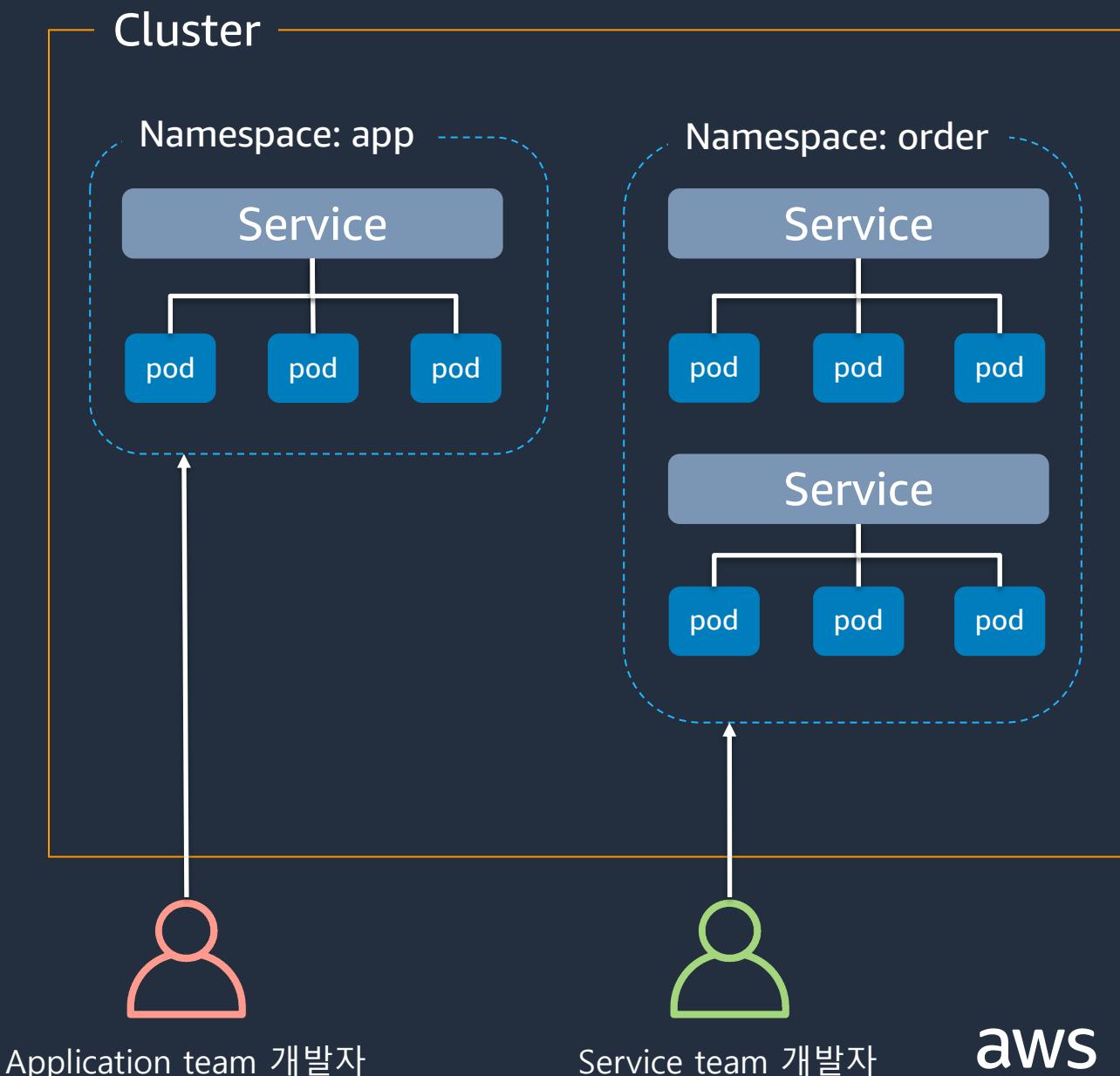


목표 : 빌드 및 풀 타임 개선 > pod의 성능과 연결

네임스페이스 잘 사용하기

- Default 네임스페이스를 사용하는 대신 운영 환경에 맞게 논리적으로 분류
 - 예: 개발/운영/테스트 환경 혹은 서비스 단위
 - ResourceQuota**를 사용하여 네임스페이스마다 리소스 할당 가능
 - RBAC을 통한 권한 분리

목표 : 관리, 보안, 그리고 성능 개선



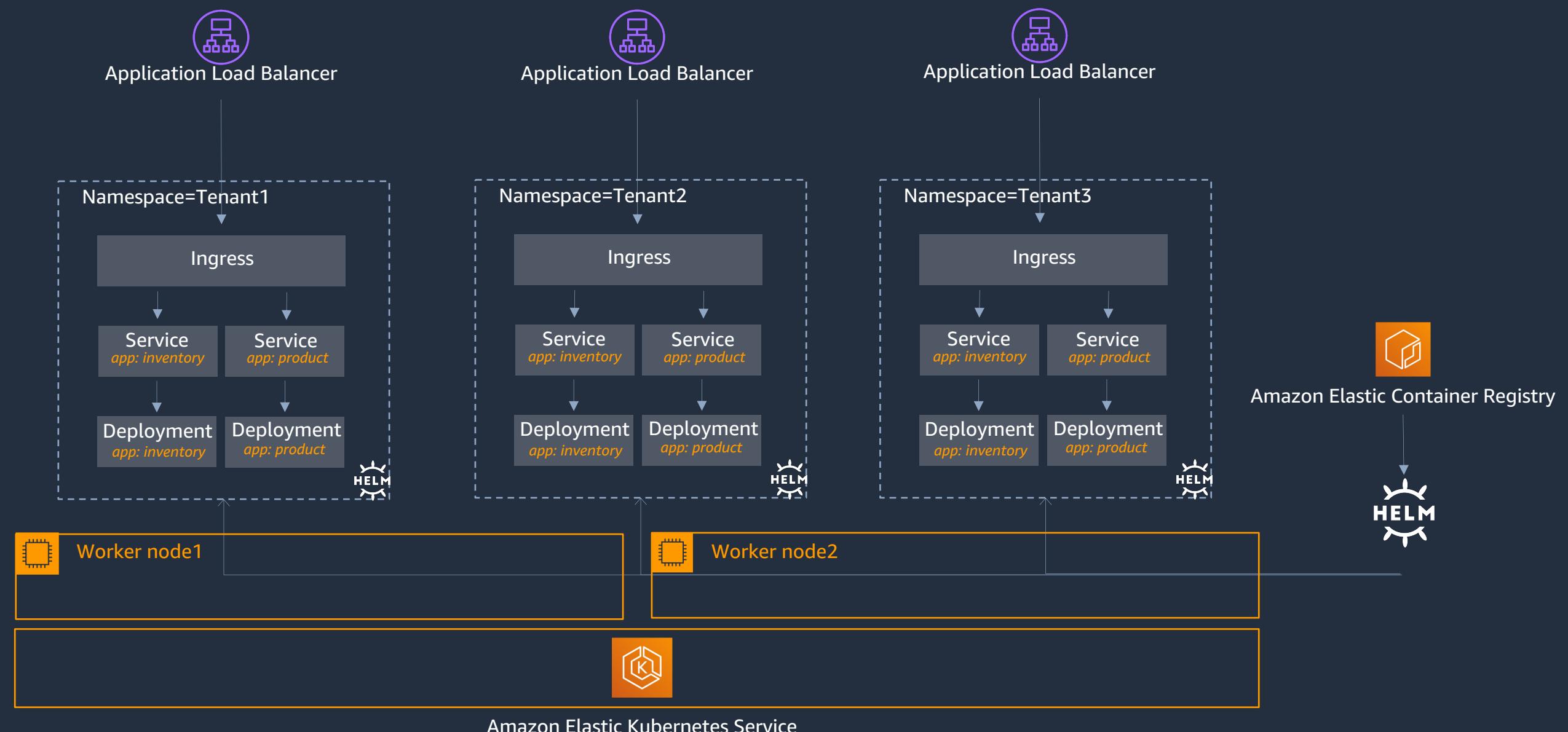
로깅 및 모니터링 다각화

- Infrastructure layer
 - Worker node resources utilization, PV, etc
- Application layer
- k8s
 - Health Check(Liveness probes , Readiness probes)
 - Yaml 파일 readinessProbe 속성에서
initialDelaySeconds: kubelet에게 해당 시간 이후에 헬스 체크를 시작하겠다고 설정하는 것

목표: 트러블 슈팅 및 신속한 장애 대응, 운영 최적화를 위한 지표 수집

Summary







감사합니다

