

Kubernetes

Contents

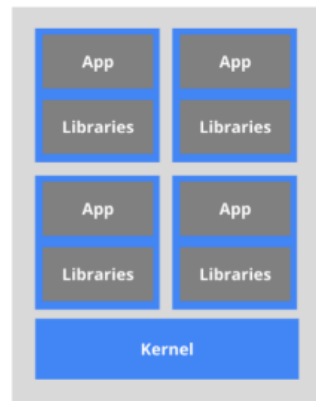
1. Understand the Basic
2. Kubernetes Installation
3. 컨테이너 배포 - Pod, Replicaset, Deployment
4. 컨테이너 통신 - Service, Ingress
5. 컨테이너 볼륨 및 환경변수 - Volume, Configmap, Secret
6. 컨테이너 관리 - Dashboard, Autoscaling

Chapter 1

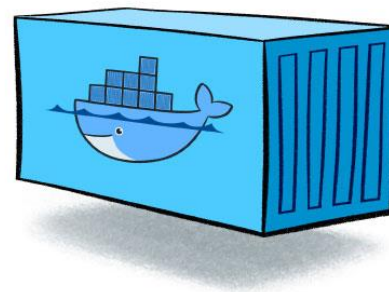
I Understand the Basic

컨테이너

- OS 가상화 기술
- 프로세스 격리
- 리눅스 커널 공유

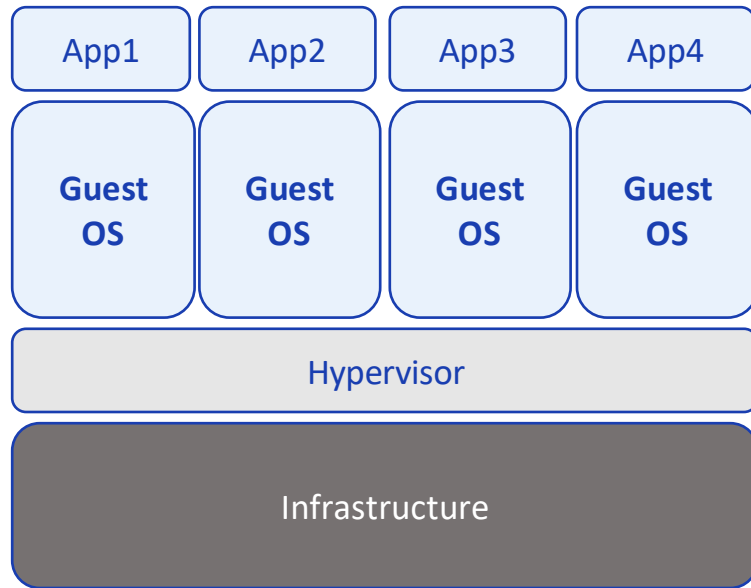


Containers

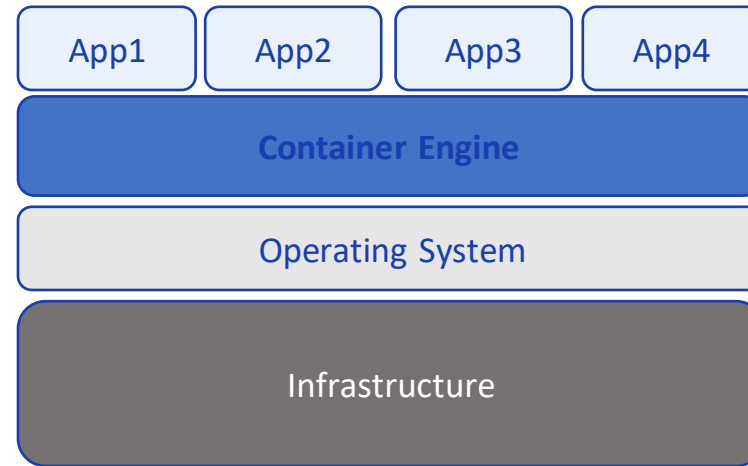


출처 : Linux Foundation, Docker Container

가상머신과 컨테이너 비교 -1



Virtual Machine



Container

출처 : Linux Foundation, Docker Container

가상머신과 컨테이너 비교 -2

| 구분 | 가상머신 | 컨테이너 |
|---------|--------------------------|---|
| 게스트OS | Windows, Linux 등 다양하게 선택 | X |
| 시작시간 | 길다(몇 분) | 짧다(몇 초) |
| 이미지 사이즈 | 크다 (수 GB ~ 수백 GB) | 작다 (~ 수백 MB) |
| 환경관리 | 각 VM마다 OS 패치가 필요 | 호스트 OS 만 패치 |
| 데이터 관리 | VM 내부 또는 연결된 스토리지에 저장 | 컨테이너 내부의 데이터는 컨테이너 종료 시 소멸, 필요시 스토리지를 이용하여 저장 |

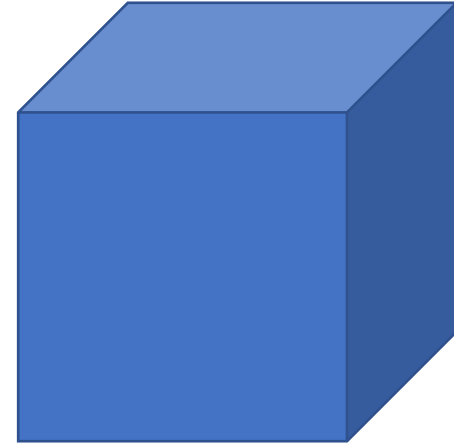
컨테이너의 장점

- ㅣ 가벼움 (Lightweight)
- ㅣ 탄력성 (Elasticity)
- ㅣ 밀집성 (Density)
- ㅣ 고성능 (Performance)
- ㅣ 효율적인 유지관리 (Maintenance Efficiency)

Monolithic vs Micro Service

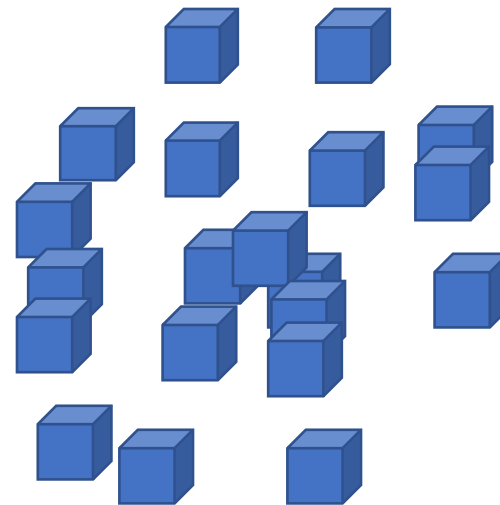
I Monolithic Architecture

- ▶ 고용량 고성능의 단일 서버로 구성



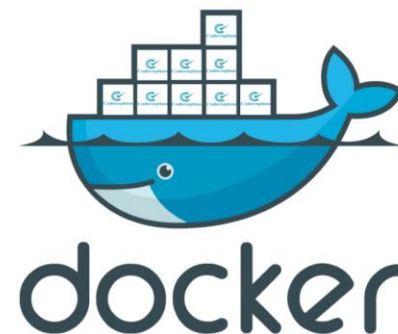
I MicroService Architecture

- ▶ Monolithic Architecture 와 비교하여
작은 서버들의 집합체로 구성



Docker

- ㅣ 컨테이너 엔진 (컨테이너를 실행하고 관리하는 도구)
- ㅣ 컨테이너 기반의 오픈소스 가상화 플랫폼
- ㅣ 도커는 도커허브라는 공개된 저장소서버를 통해,
컨테이너 자료들을 관리합니다.
- ㅣ 컨테이너를 생성하고 실행하기 위해서는
Dockerfile과 Image가 필요합니다.



Dockerfile

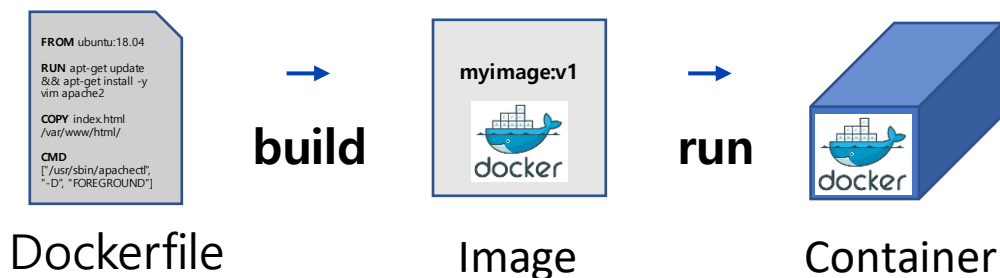
- 컨테이너 이미지를 생성하기 위한 레시피 파일입니다.
- 이 파일에 이미지 생성과정을 문법에 따라 작성하여 저장합니다
- FROM, WORKDIR, RUN, CMD 등
용도에 따른 명령어 모음

```
FROM ubuntu:18.04
```

```
RUN apt-get update && apt-get install -y vim apache2
```

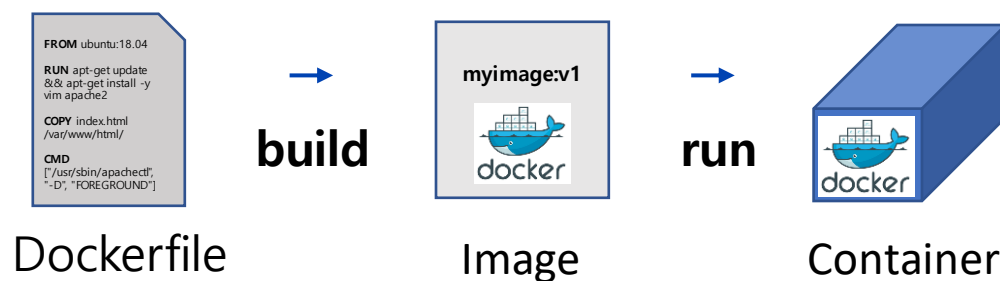
```
COPY index.html /var/www/html/
```

```
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```



Docker Image

- 서비스 운영에 필요한 프로그램, 소스코드, 라이브러리 등을 묶는 형태
- 도커 이미지는 Dockerfile을 사용하여 생성할 수 있습니다.
(Build)
- 도커 이미지를 사용하여 다수의 Container를 실행할 수 있습니다. (Run)



Docker Image 경로

- 도커 이미지는 url 방식으로 관리하고, 태그를 붙일 수 있습니다.

- 도커 이미지의 형식

 - ▶ <Namespace>/<ImageName>:<Tag>

docker.io/library/nginx:latest

nginx:latest

nginx

Docker Image 경로 (Private)


- 또한 private 한 이미지저장소를 구축하여 이미지를 관리한다면, <Namespace>부분을 해당 서버주소 및 포트번호 등으로 사용할 수 있습니다.
- <Namespace>/<ImageName>:<Tag>
private:10000/mynginx:latest

Docker HUB

- | 수많은 컨테이너 이미지들을 서버에 저장하고 관리
- | 공개 이미지를 무료로 관리
- | <https://hub.docker.com/>



Docker HUB 이미지 목록



[Explore](#) [Pricing](#) [Sign In](#) [Sign Up](#)

[Docker](#) [Containers](#) [Plugins](#)

Filters (1) [Clear All](#)

Images

☐ Verified Publisher i

☒ Official Images i
Official Images Published By Docker

Categories i

☐ Analytics

☐ Application Frameworks

☐ Application Infrastructure

☐ Application Services

☐ Base Images

☐ Databases

☐ DevOps Tools

☐ Featured Images

☐ Messaging Services


☐ Monitoring

☐ Operating Systems

☐ ...

1 - 25 of 172 available images.

Suggested



ubuntu

Official Image

Updated a month ago

Ubuntu is a Debian-based Linux operating system based on free software.

Container

Linux

ARM 64

ARM

IBM Z

riscv64


x86-64

386

PowerPC 64 LE

Base Images

Operating Systems



redis

Official Image

Updated 14 days ago

Redis is an open source key-value store that functions as a data structure server.

Container

Linux

Windows

mips64le

x86-64

ARM


PowerPC 64 LE

IBM Z

ARM 64

386

Databases



alpine

Official Image

Updated a month ago

1B+10K+

DownloadsStars


1B+10K+

DownloadsStars

1B+8.3K

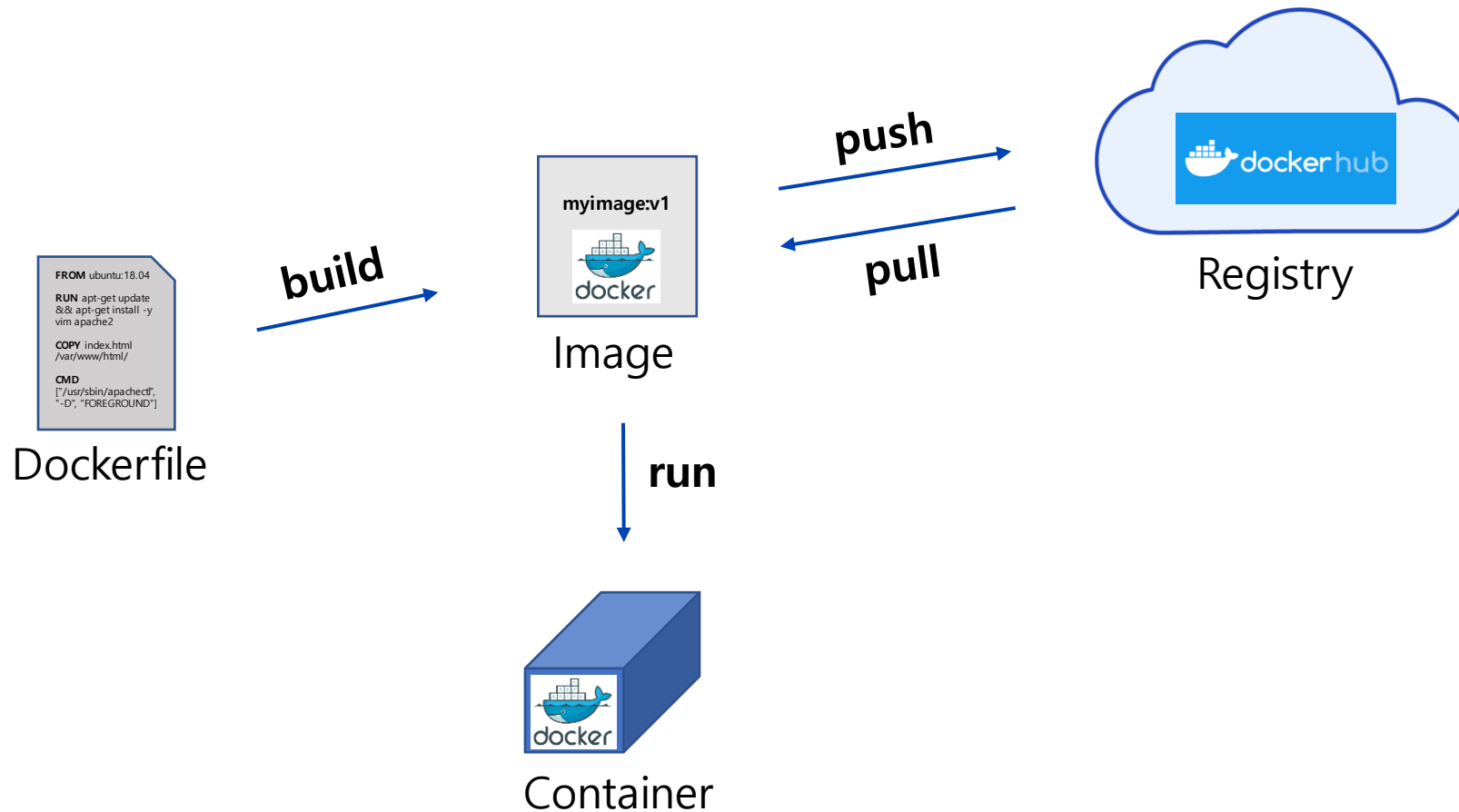
DownloadsStars

Copyright (c) 2022 Global Knowledge Korea Training LLC All rights reserved.

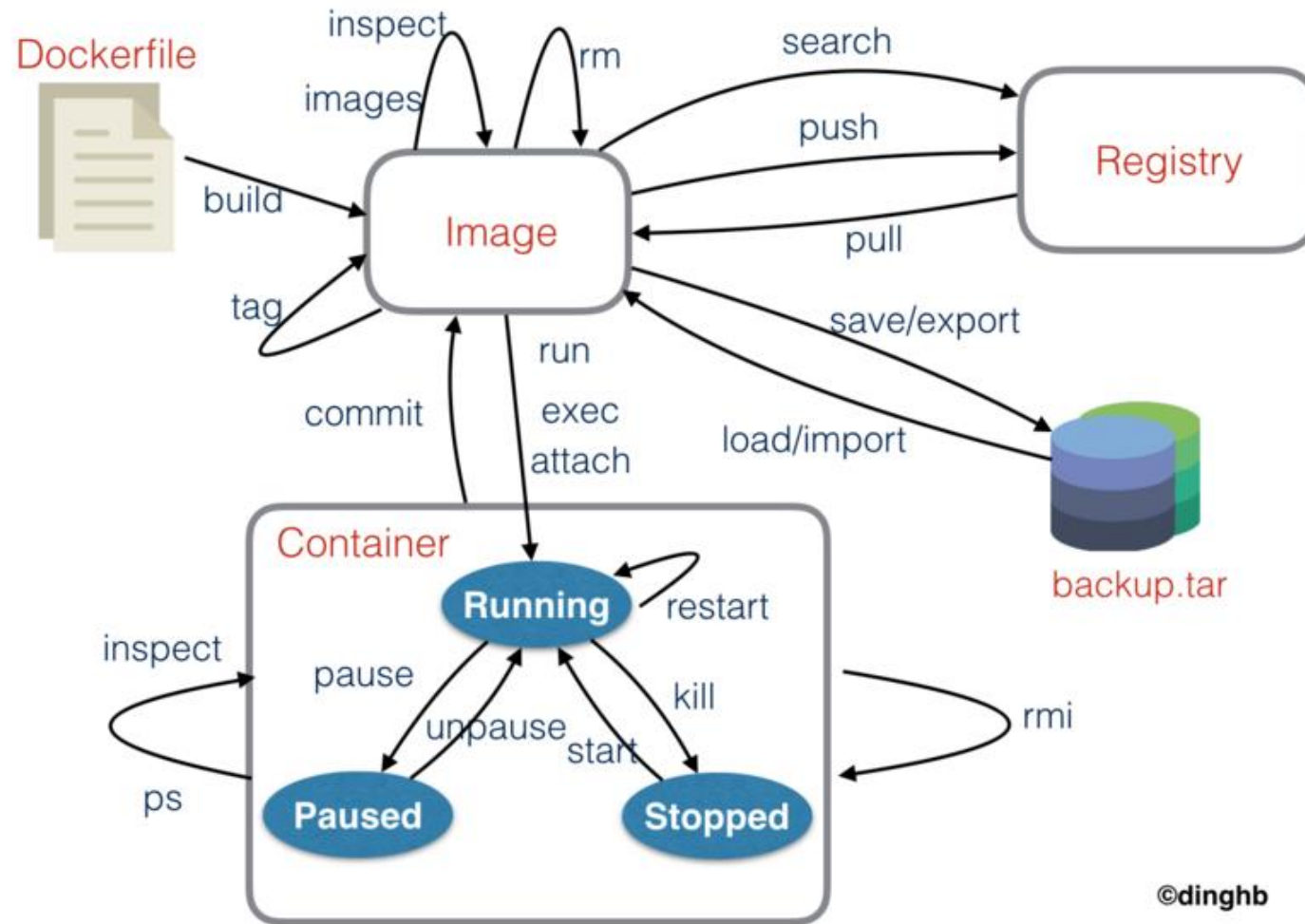


Global Knowledge

Docker 간단 명령



Docker 명령 구조



컨테이너 오케스트레이션

- 다수의 컨테이너를, 다수의 시스템에서, 각각의 목적에 따라, 배포/복제/장애복구 등 총괄적으로 관리하는 것.
- 컨테이너 오케스트레이션 도구들의 일반적 기능
 - ▶ 스케줄링
 - ▶ 자동확장 및 축소
 - ▶ 장애복구
 - ▶ 로깅 및 모니터링
 - ▶ 검색 및 통신
 - ▶ 업데이트 및 롤백

컨테이너 오케스트레이터

- ㅣ 컨테이너 오케스트레이션을 해주는 도구
- ㅣ 컨테이너 오케스트레이터의 종류
 - ▶ Kubernetes
 - ▶ Docker Swarm
 - ▶ AWS ECS
 - ▶ Azure Container Instance
 - ▶ Azure Service Fabric
 - ▶ Marathon
 - ▶ Nomad

컨테이너 오케스트레이터의 배포위치

I 컨테이너 오케스트레이터의 배포 위치

- ▶ 베어 메탈
- ▶ 가상머신
- ▶ 온프레미스
- ▶ 클라우드

Kubernetes

- 컨테이너형 애플리케이션의 배포, 확장, 관리를 자동화하는 오픈 소스 시스템



출처: Kubernetes.io

Kubernetes의 기능

I 쿠버네티스의 기능

- ▶ Automatic bin packing
- ▶ Self-healing
- ▶ Horizontal scaling
- ▶ Service discovery and Load balancing
- ▶ Automated rollouts and rollbacks
- ▶ Secret and configuration management
- ▶ Storage orchestration
- ▶ Batch execution

Kubernetes를 사용하는 이유

- 높은 확장성, 원활한 이동성(이식성)
- 퍼블릭/프라이빗/하이브리드/멀티 클라우드, 로컬 또는 원격 가상 머신, 베어메탈과 같은 여러 환경에 구축 가능
- 오픈 소스 도구의 장점
- 플러그가 가능한 모듈 형식

Kubernetes 도입 기업

I 쿠버네티스를 도입하여 사용하고있는 기업들



출처: Kubernetes.io

CNCF(Cloud Native Computing Foundation)

- 대표적으로 Kubernetes 와 Prometheus 와 같은 클라우드 네이티브 오픈소스 기술들을 추진하고 관리하는 단체



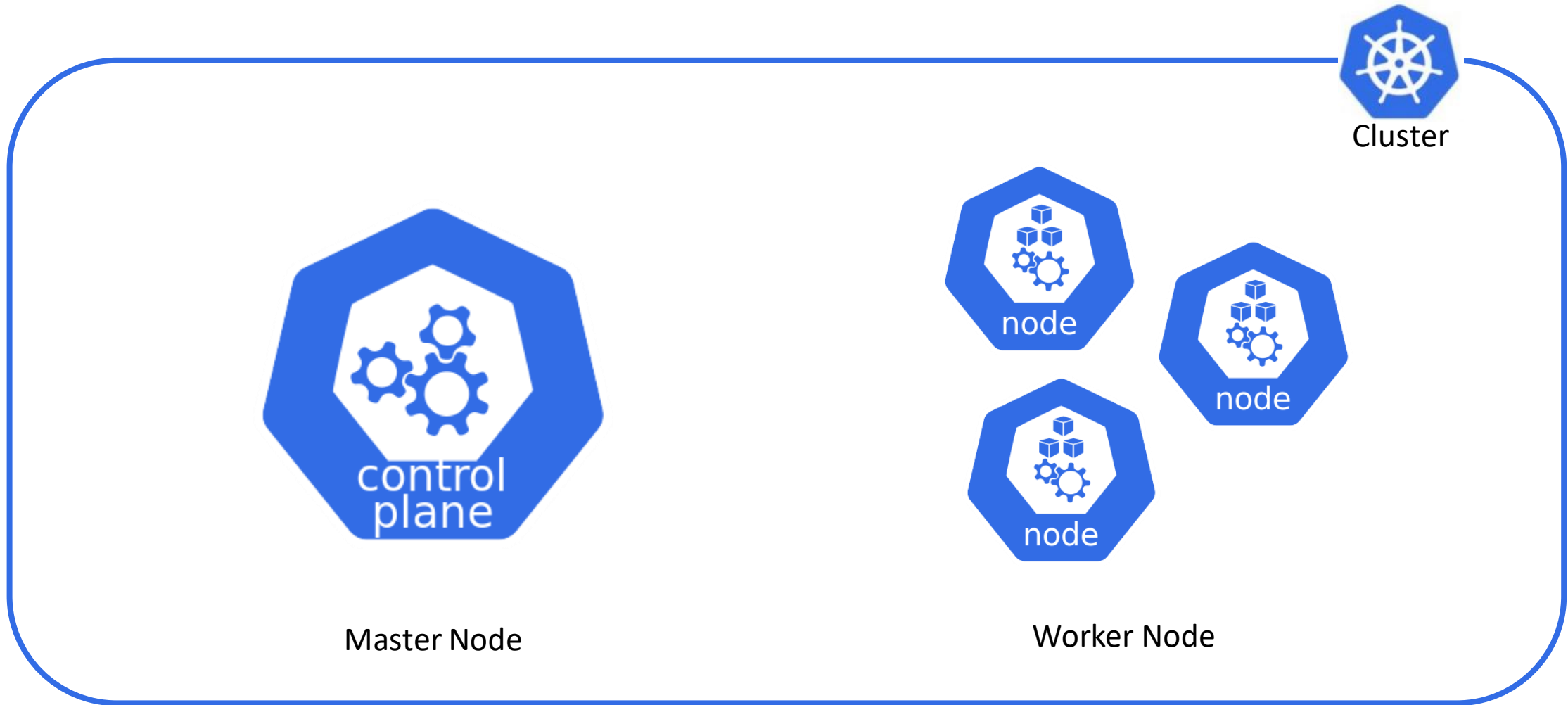
CLOUD NATIVE
COMPUTING FOUNDATION

출처: Kubernetes.io

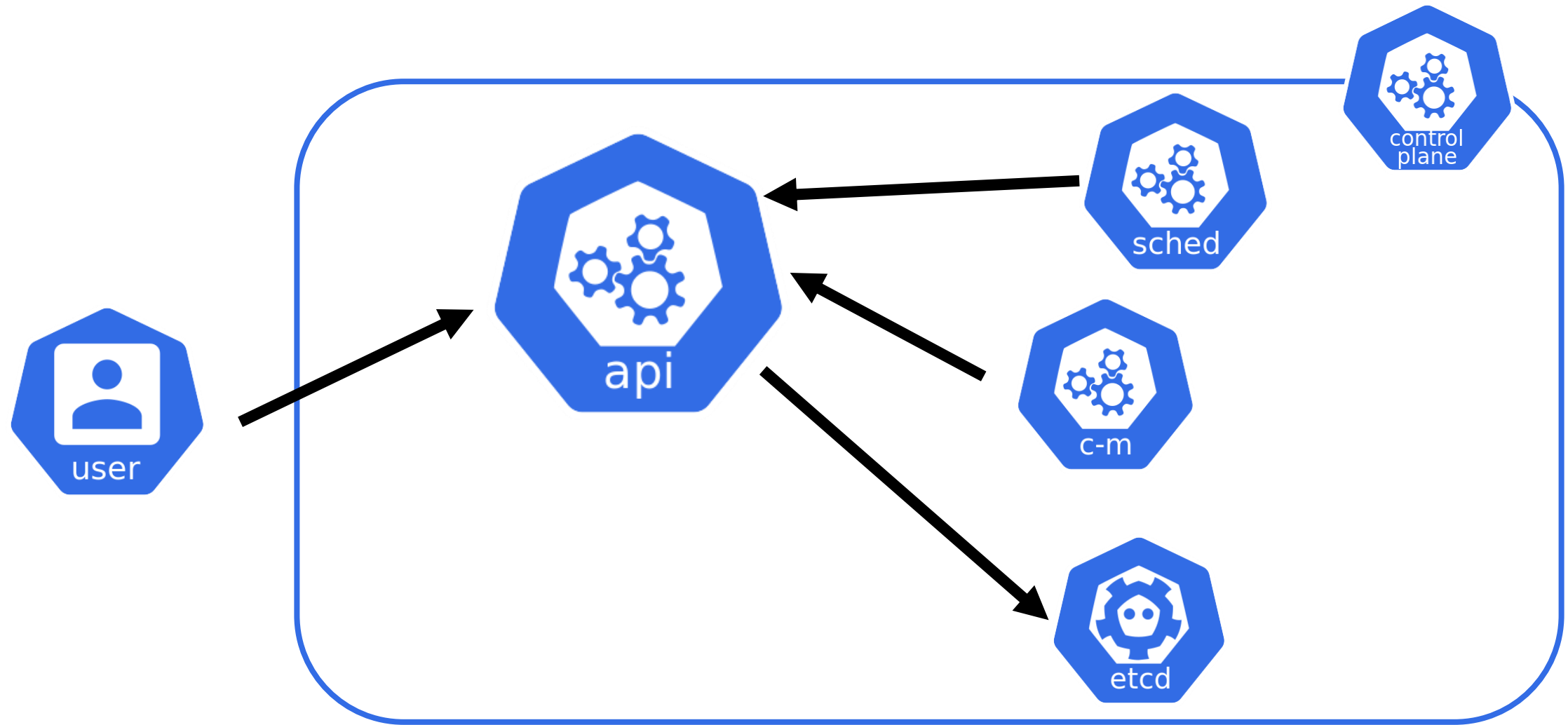
CNCF의 완료 프로젝트

- | Kubernetes for container orchestration
- | Prometheus for monitoring
- | Envoy for service mesh
- | CoreDNS for service discovery
- | containerd for container runtime
- | Fluentd for logging

Kubernetes 아키텍처

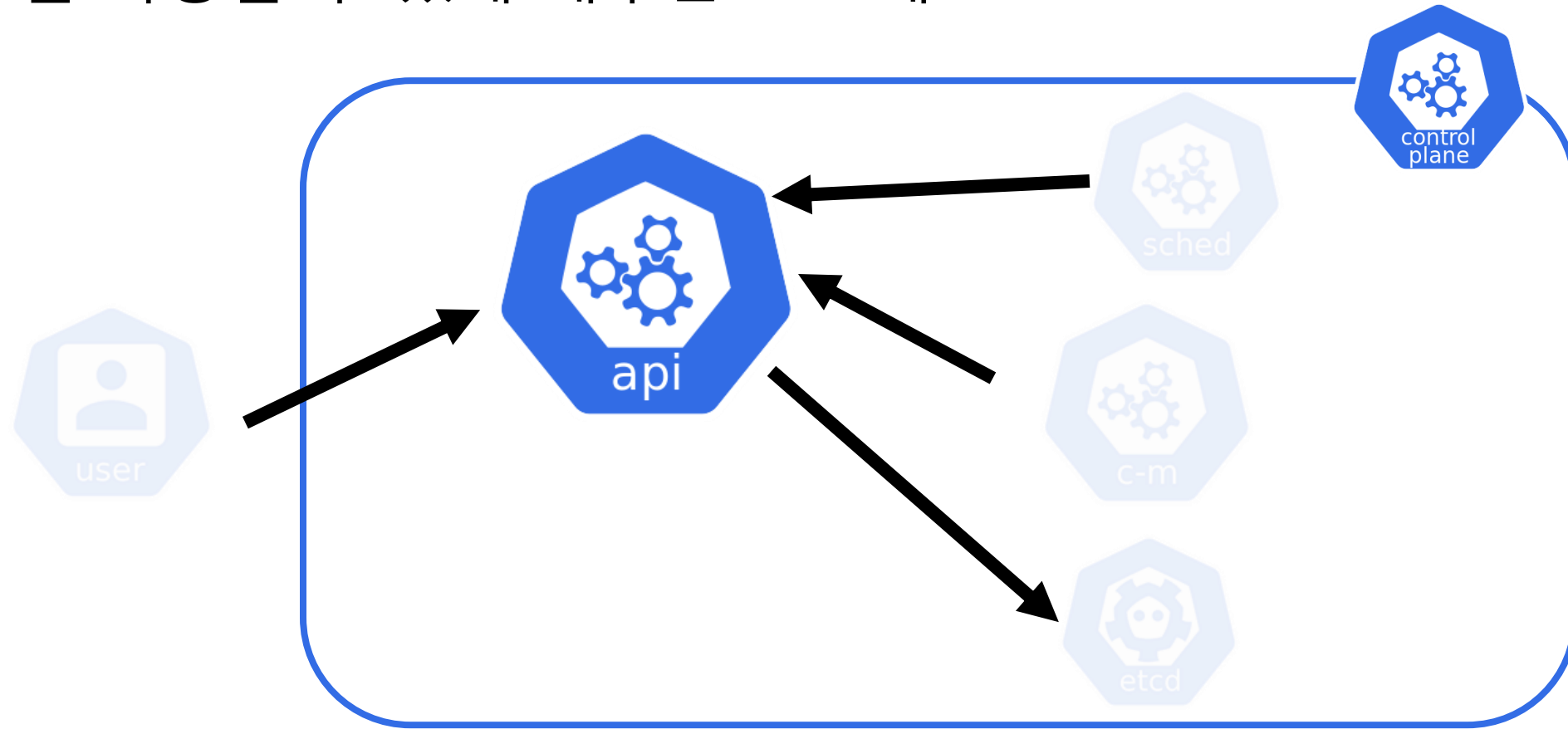


마스터 노드의 구성요소



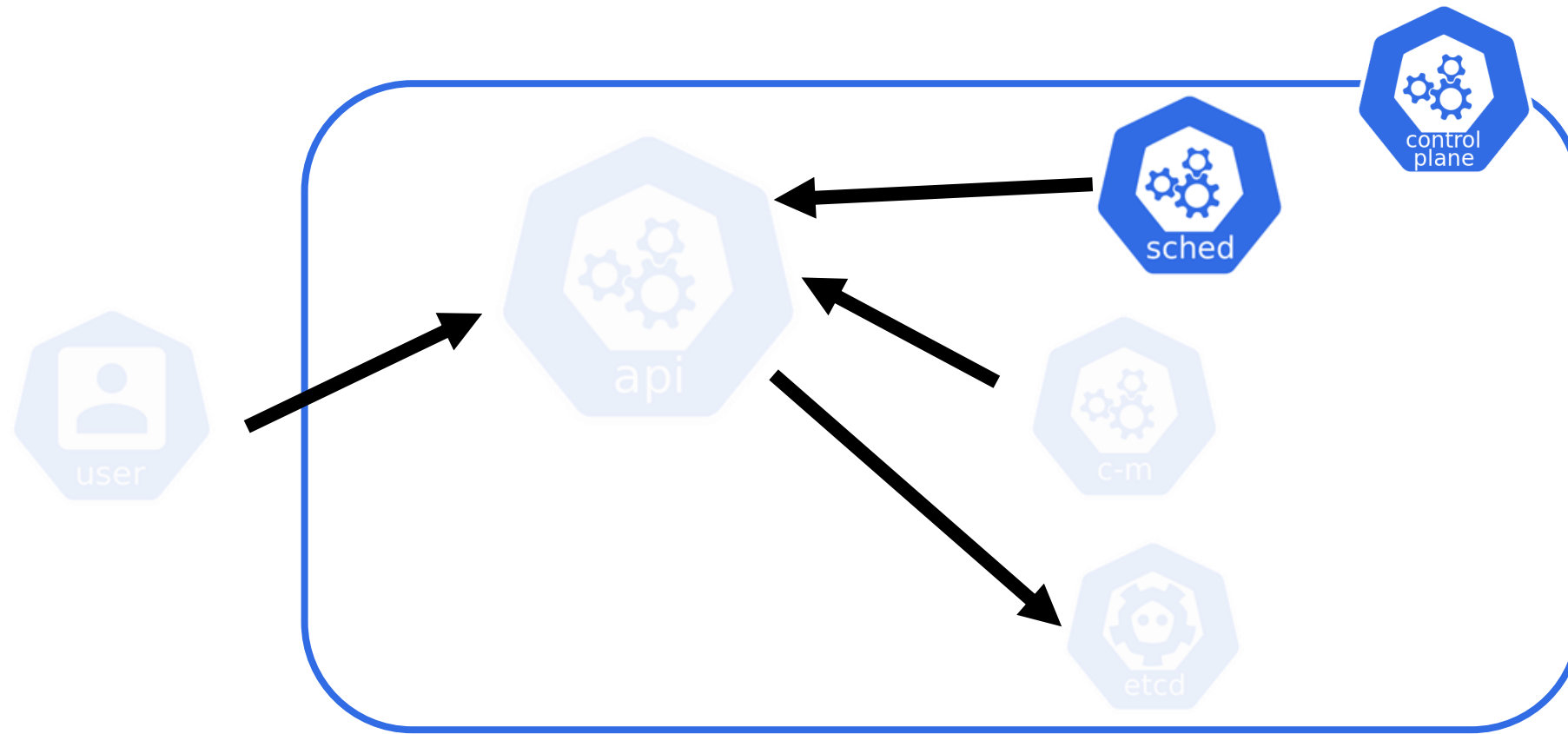
API Server

I API를 사용할 수 있게 해주는 프로세스



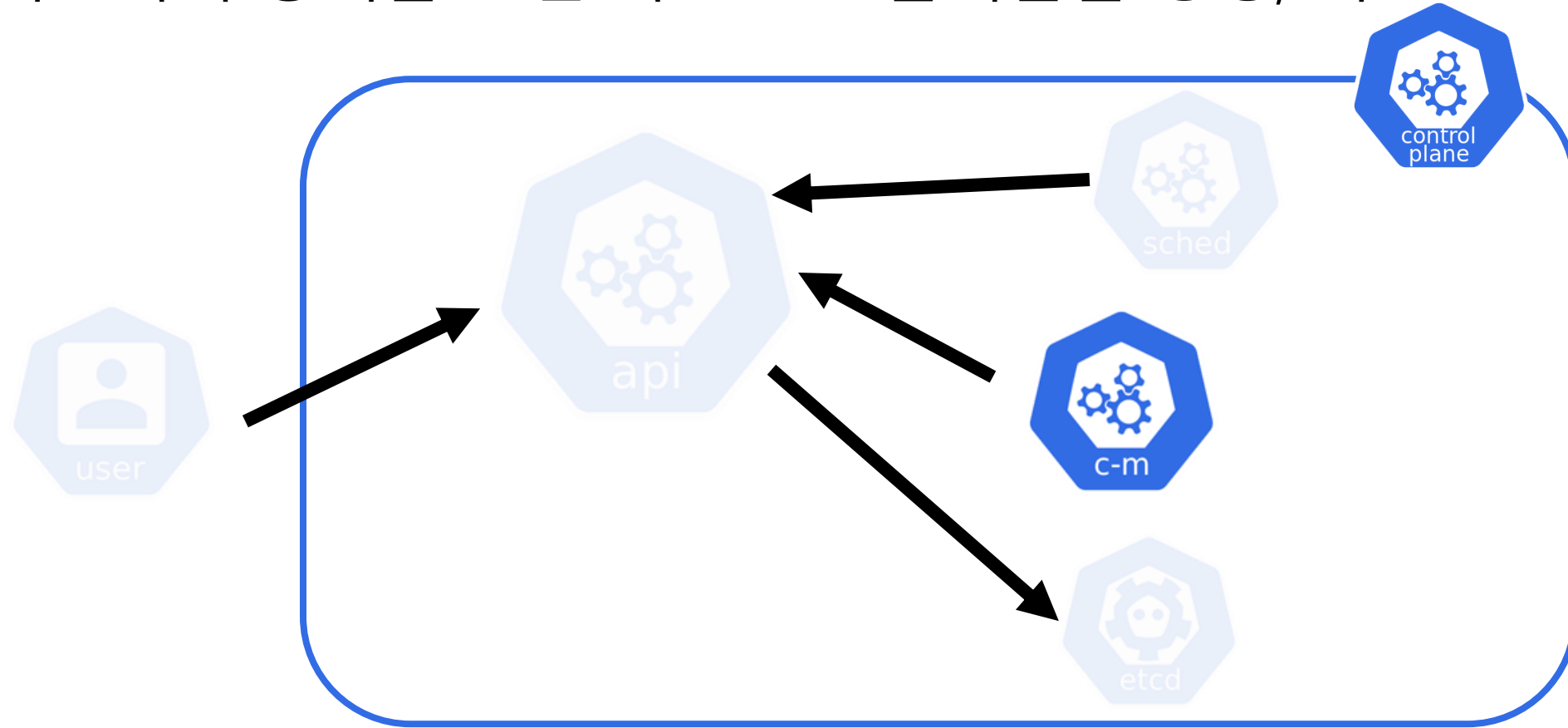
Scheduler

- Pod의 생성 명령이 있을경우 어떤 Node에 배포할지 결정



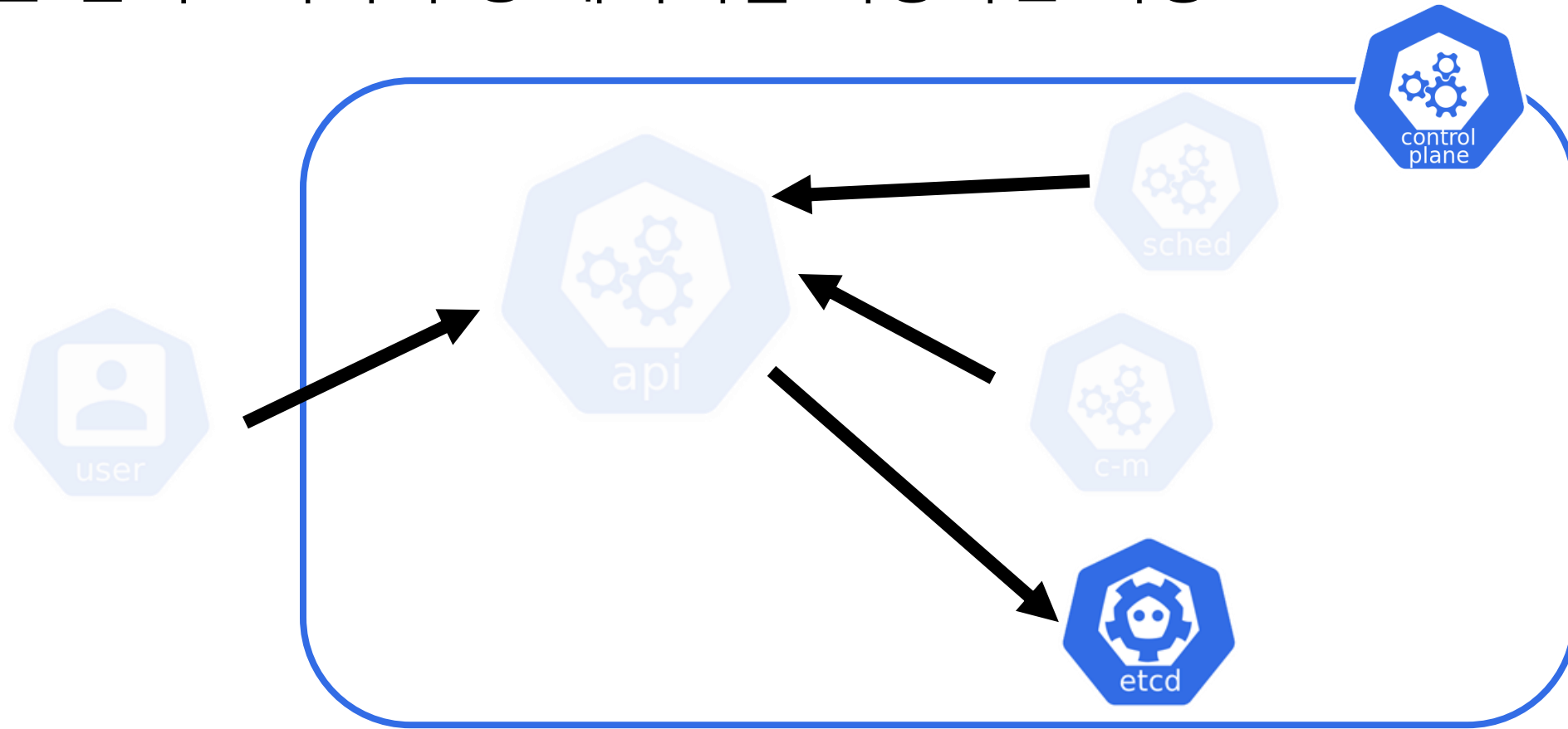
Controller Managers

클러스터의 상태를 조절 하는 컨트롤러들을 생성, 배포



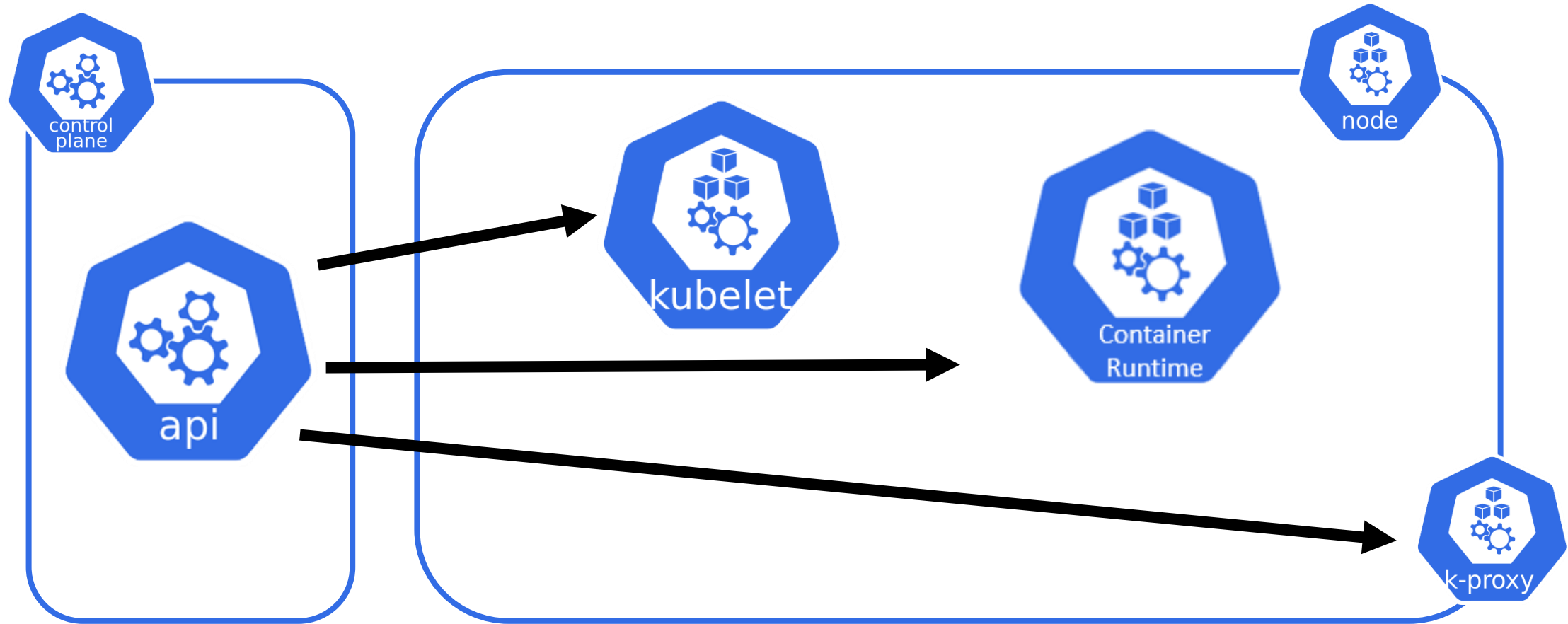
etcd

I 모든 클러스터의 구성 데이터를 저장하는 저장소



워커 노드의 구성요소

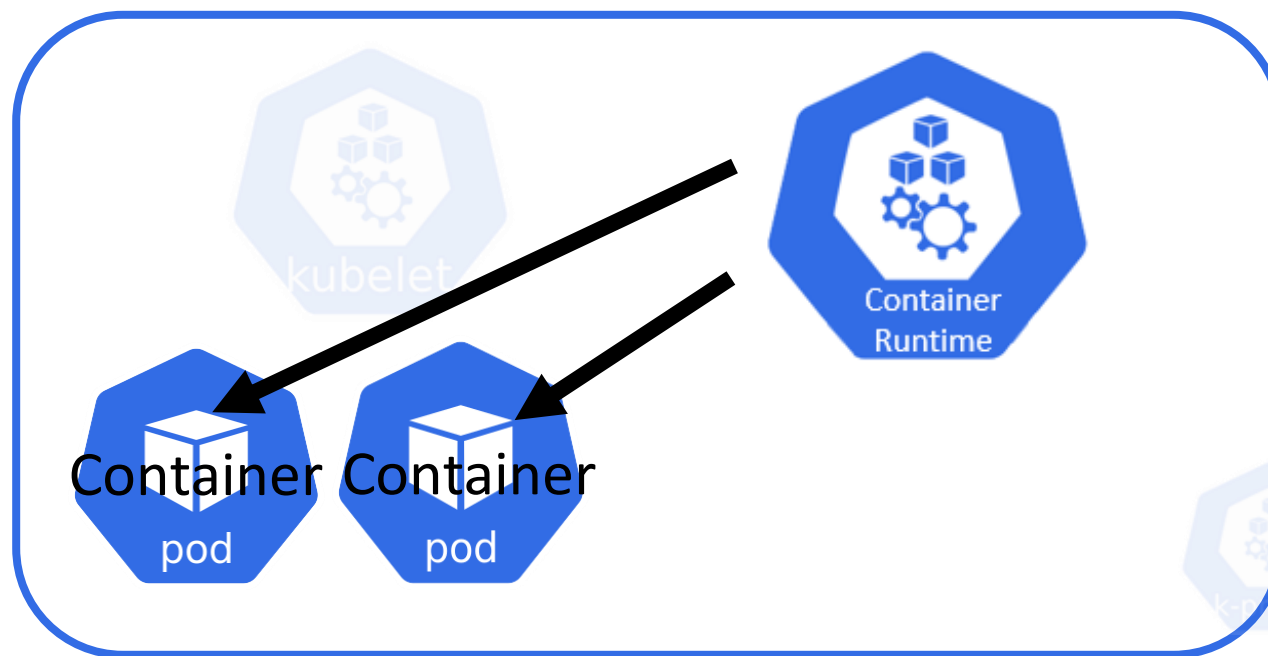
I 컨테이너가 배포될 워커머신



컨테이너 Runtime

- ㅣ 컨테이너를 실행하고 노드에서 컨테이너 이미지를 관리합니다.
- ㅣ Kubernetes에서 지원하는 컨테이너 Runtime은 다음과 같습니다.

- ▶ 도커
- ▶ CRI-O
- ▶ containerd
- ▶ rkt
- ▶ rktlet



kubelet

I 각 Node의 에이전트

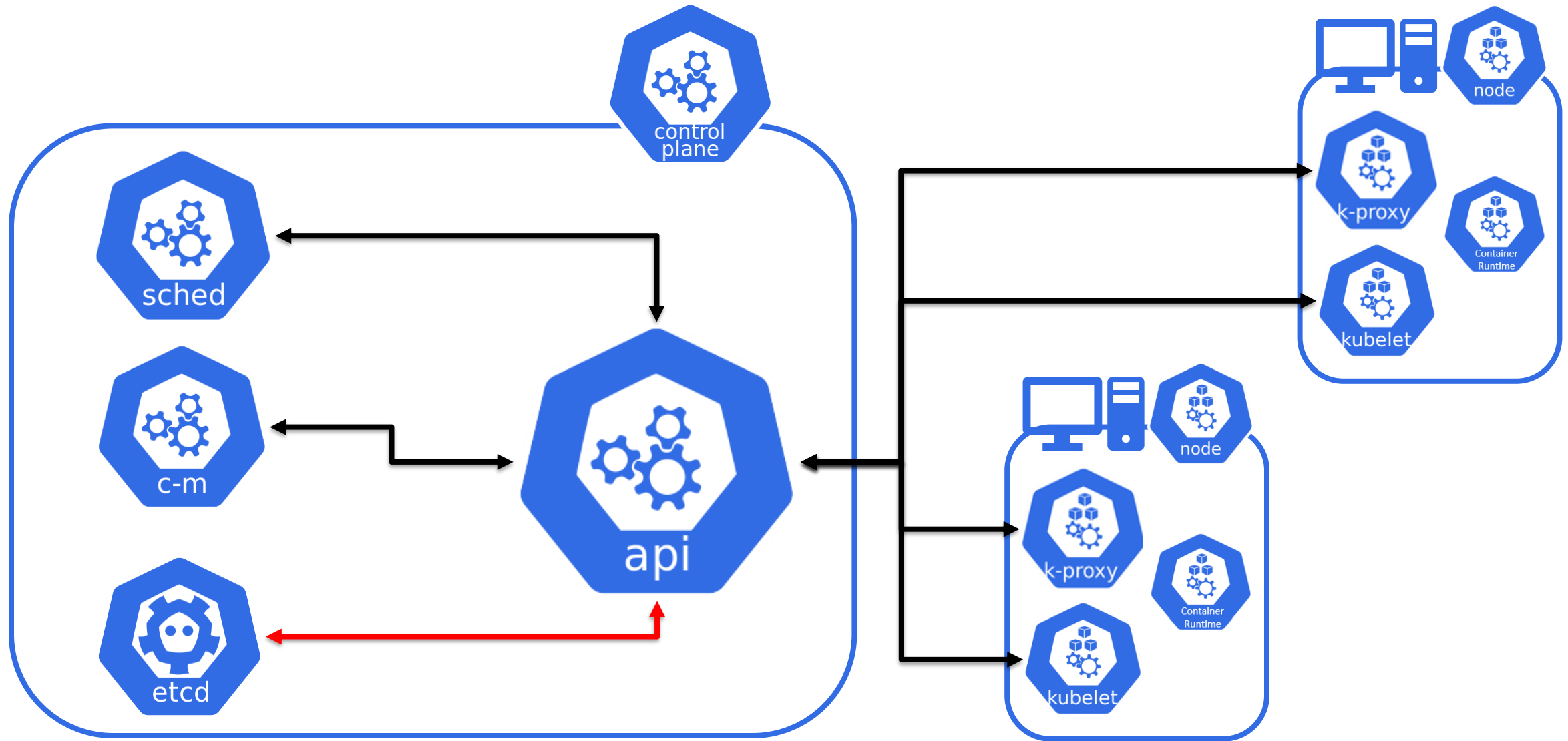


kube-proxy

- 각 노드의 네트워크 규칙 유지 관리
- 각 노드 간 통신을 담당



Kubernetes 아키텍처



Addons

- | Addons 은 Kubernetes에서 추가적으로 설치하여 Kubernetes의 기능을 확장 시킬 수 있는 도구입니다.
- | Kubernetes Addons의 종류는 다음과 같습니다.
 - ▶ DNS
 - ▶ Dashboard
 - ▶ Monitoring
 - ▶ Logging
 - ▶ Etc...

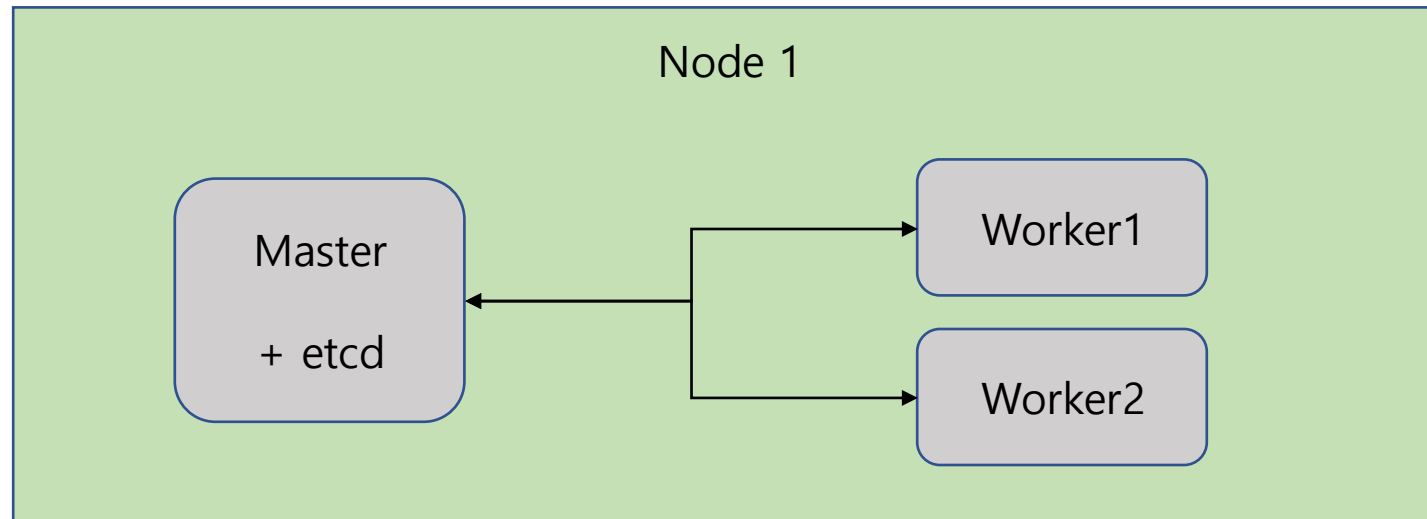
Chapter 2

I Kubernetes Installation

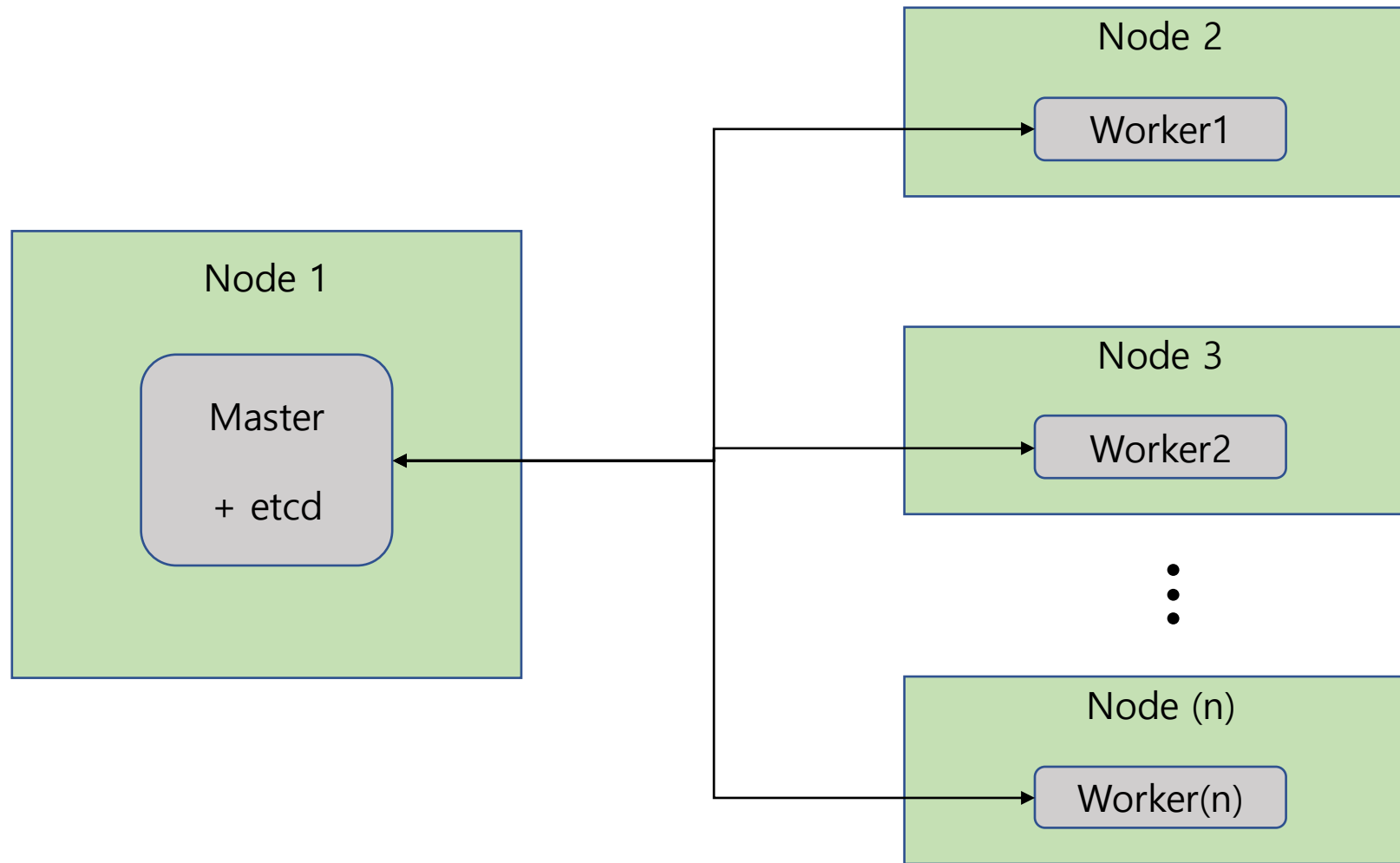
Kubernetes 설치 유형

- | All-in-One Single-Node Installation
- | Single-Node etcd, Single-Master and Multi-Worker Installation
- | Single-Node etcd, Multi-Master and Multi-Worker Installation
- | Multi-Node etcd, Multi-Master and Multi-Worker Installation

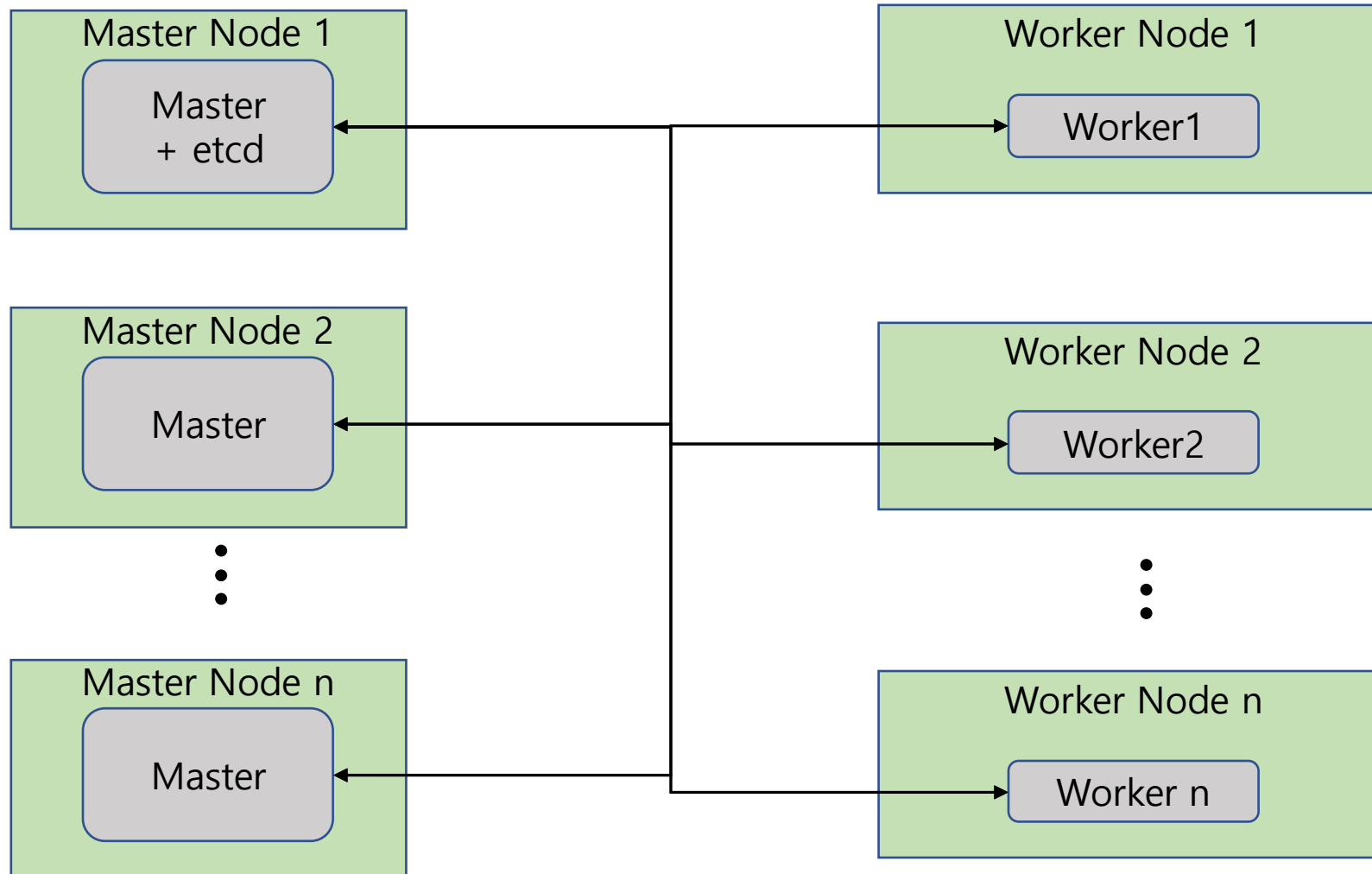
All-in-One Single-Node Installation



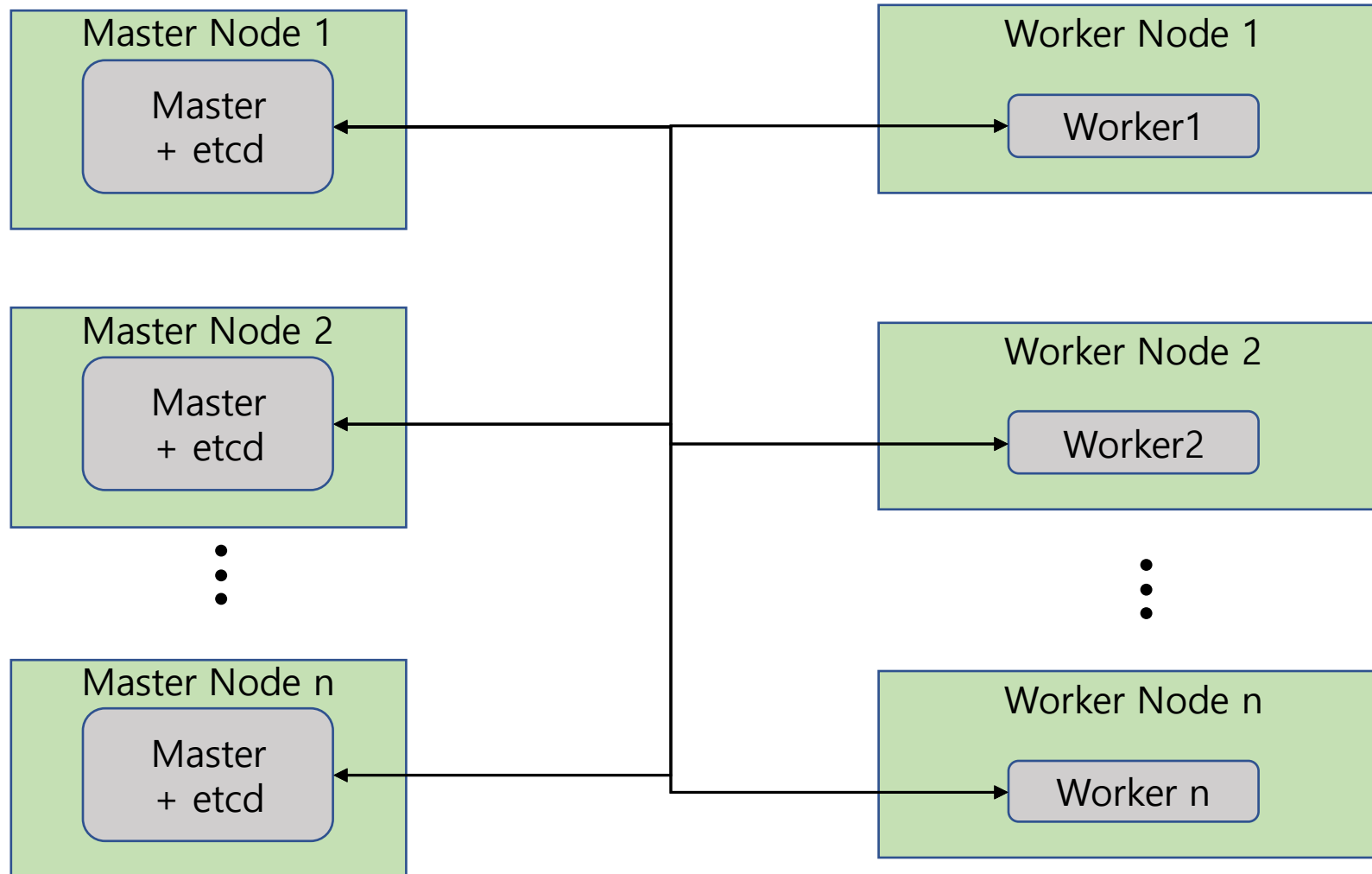
Single-Node etcd, Single-Master and Multi-Worker Installation



Single-Node etcd, Multi-Master and Multi-Worker Installation



Multi-Node etcd, Multi-Master and Multi-Worker Installation



Kubernetes 설치 도구

- | kubeadm
- | kubespray
- | kops

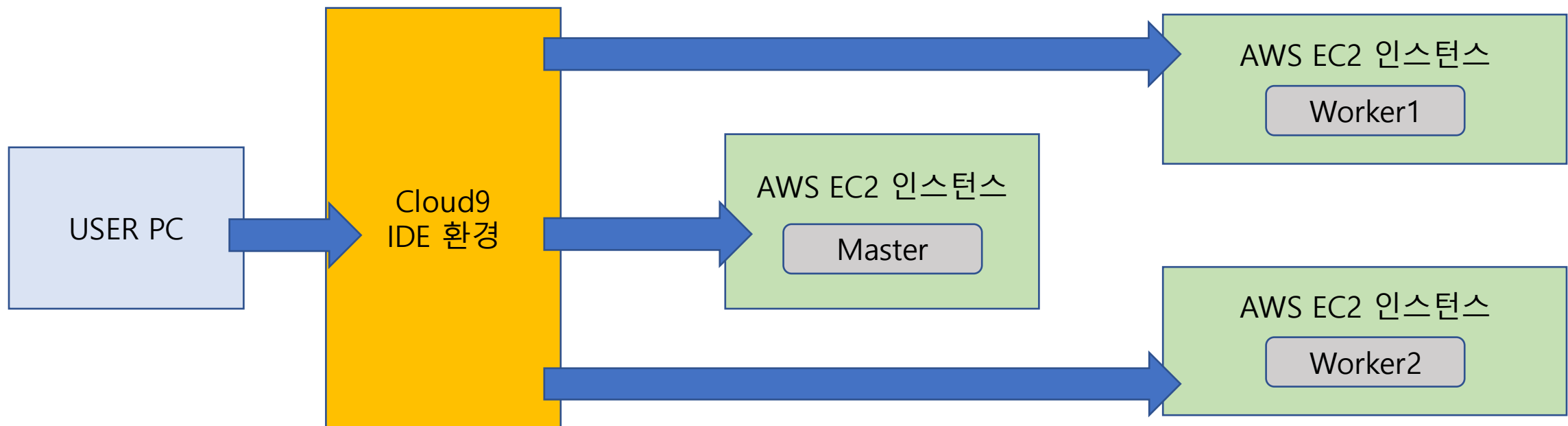


Kubernetes 설치 순서

- | Docker 설치
- | Kubernetes 설치
- | Master 와 Worker 연동

실습 환경 안내

- | AWS 의 Cloud9 서비스 사용
- | AWS 의 EC2 인스턴스 서비스 사용



Chapter 3

I 컨테이너 배포 - Pod, Replicaset, Deployment

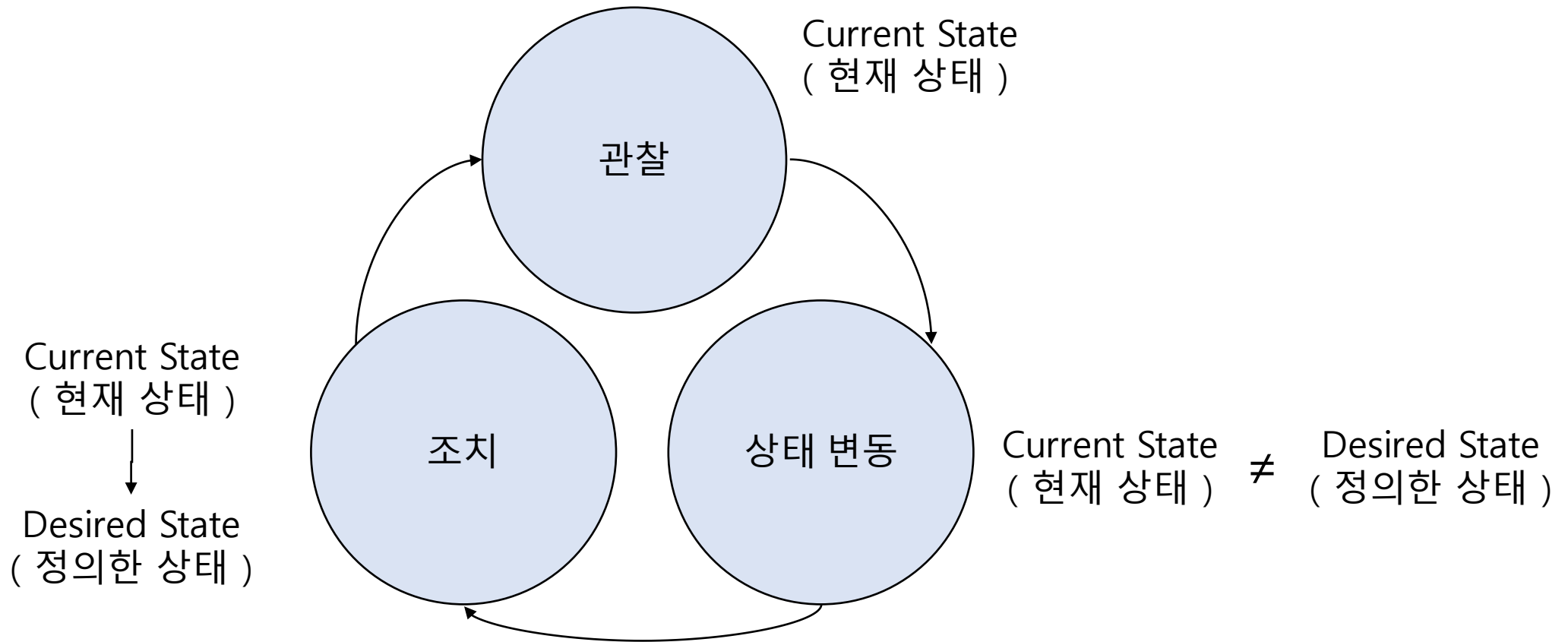
Kubernetes Object

- ㅣ 가장 기본적인 구성단위
- ㅣ 상태를 관리하는 역할
- ㅣ 가장 기본적인 오브젝트
 - ▶ Pod, Service, Volume, Namespace
- ㅣ 오브젝트의 Spec, Status 필드
 - ▶ Spec : 정의된 상태
 - ▶ Status : 현재 상태

Kubernetes Controller

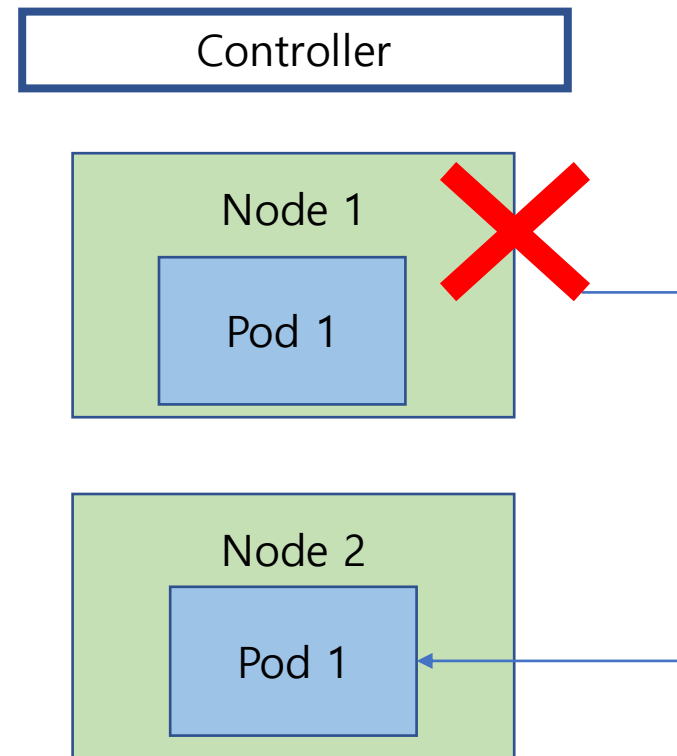
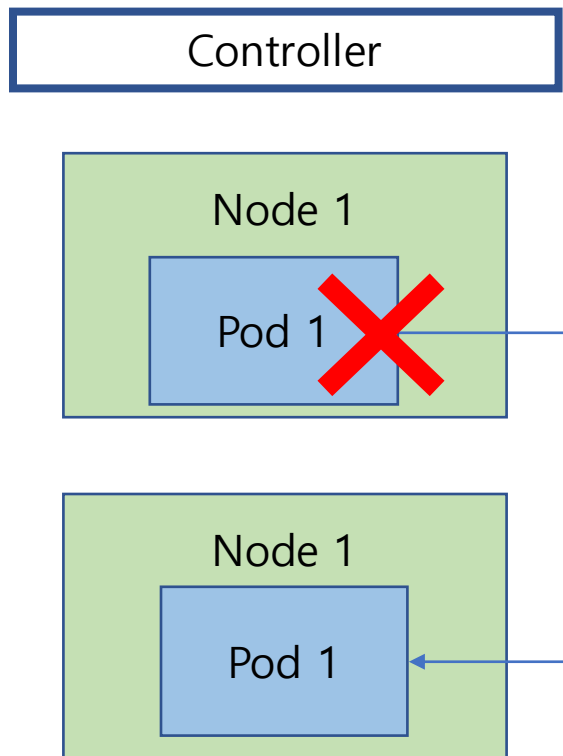
- | 클러스터의 상태를 관찰하고, 필요한 경우에 오브젝트를 생성, 변경을 요청하는 역할
- | 각 컨트롤러는 현재 상태를 정의된 상태에 가깝게 유지하려는 특징
- | 컨트롤러 유형 :
 - ▶ Deployment, Replicaset, Daemonset, Job, CronJob 등

Current State & Desired State



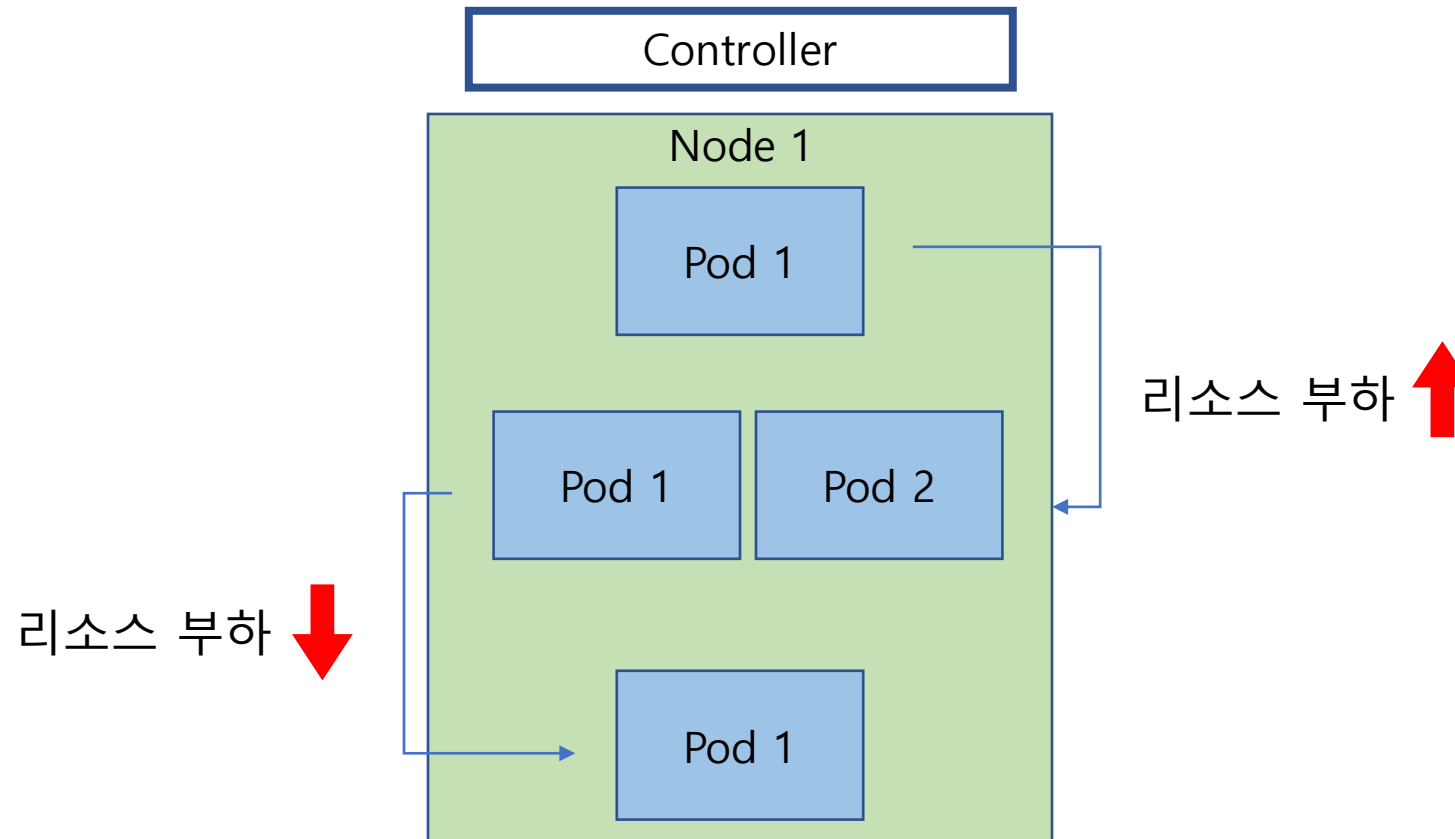
Controller 기능

Auto Healing



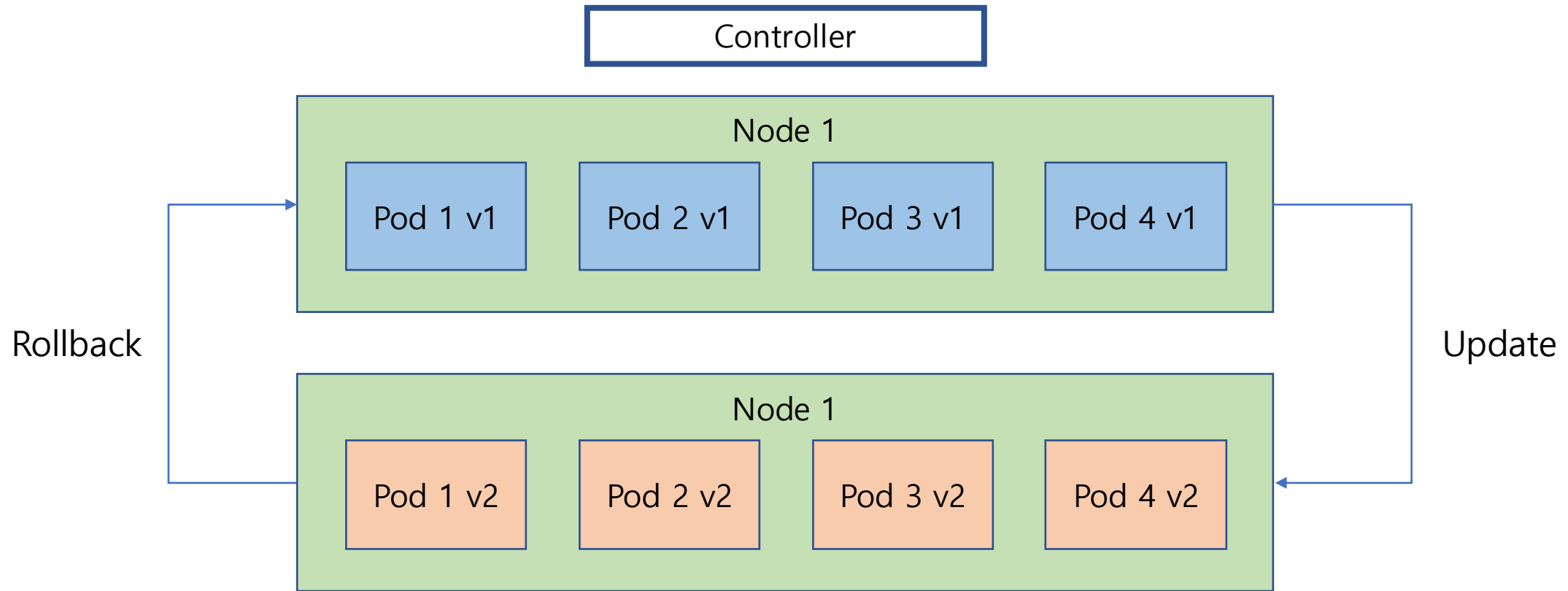
Controller 기능

I Auto Scaling



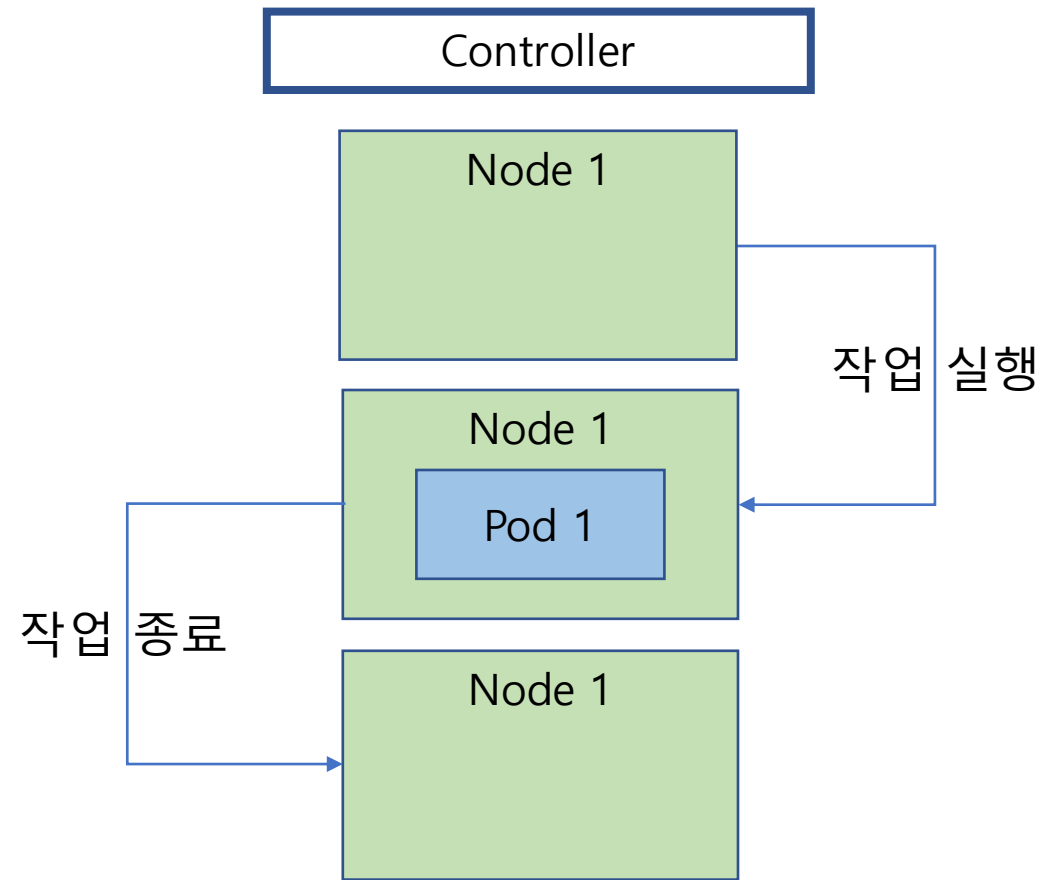
Controller 기능

Update & Rollback



Controller 기능

I Job



I Object Model

- ▶ apiVersion : 연결할 API server의 버전
- ▶ kind : Object의 유형
- ▶ metadata : Object의 기본 정보를 갖고있는 필드
name, label, namespace 등
- ▶ spec : 배포되는 Object의 원하는 상태

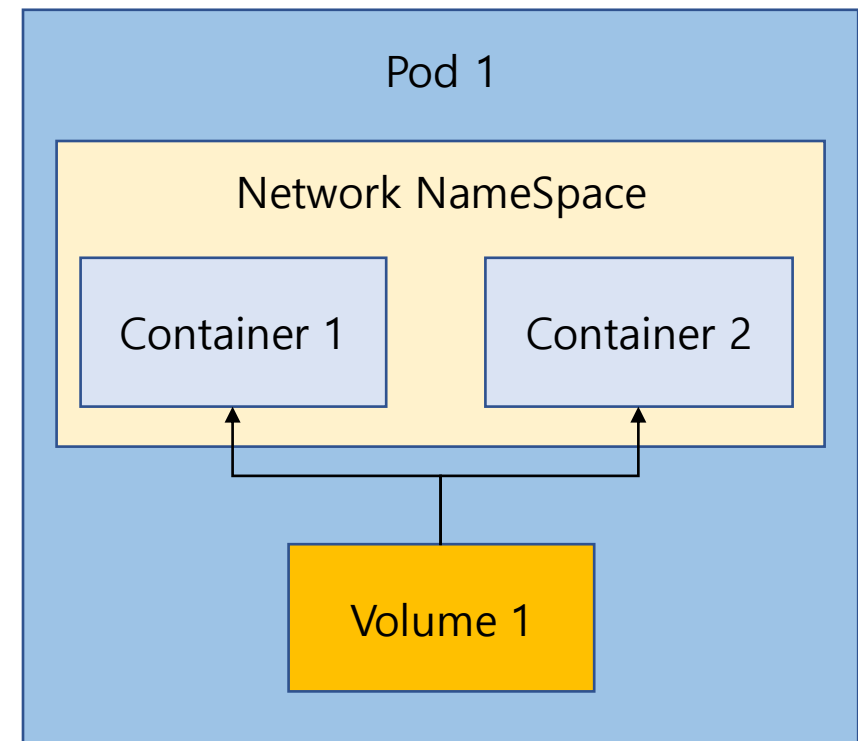
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.15.11
    ports:
    - containerPort: 80
```


Kubectl

- | Kubernetes에 명령을 내리는 CLI(Command Line Interface)
- | kubectl 명령 구조 :
 - ▶ kubectl [COMMAND] [TYPE] [NAME] [FLAGS]
- | 오브젝트와 컨트롤러를 생성, 수정, 삭제

Kubernetes Object - Pod

- Kubernetes의 가장 작은, 최소 단위 Object
- 하나 이상의 컨테이너 그룹
- 네트워크와 볼륨을 공유



Pod yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.0
    ports:
    - containerPort: 80
```

Pod - Kubectl

I yaml 파일을 사용하여 Pod를 생성 하는 명령어

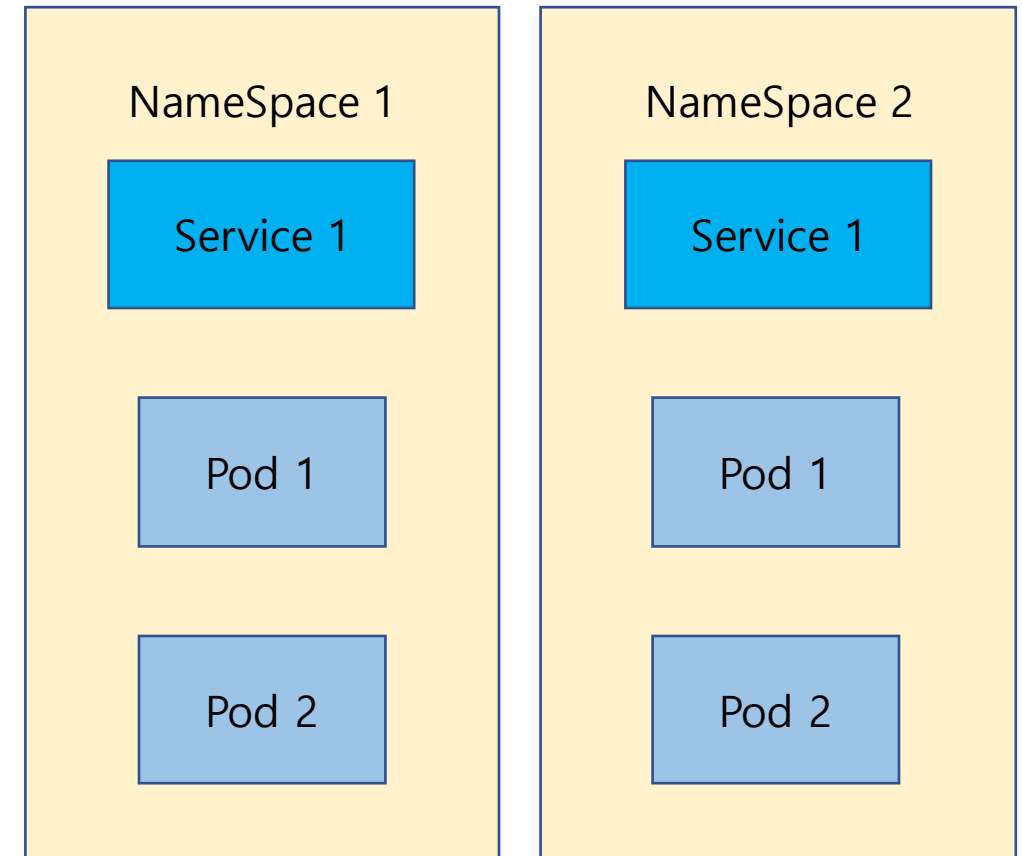
- ▶ `kubectl create -f pod.yaml`
- ▶ `kubectl create -f <yaml 파일명>`

I kubectl 명령으로 Pod를 생성하는 명령어

- ▶ `kubectl run pod --image=nginx:1.14.0 --port=80`
- ▶ `kubectl run <pod명> --image=<이미지명:버전> --port=<포트번호>`

Kubernetes Object - Namespace

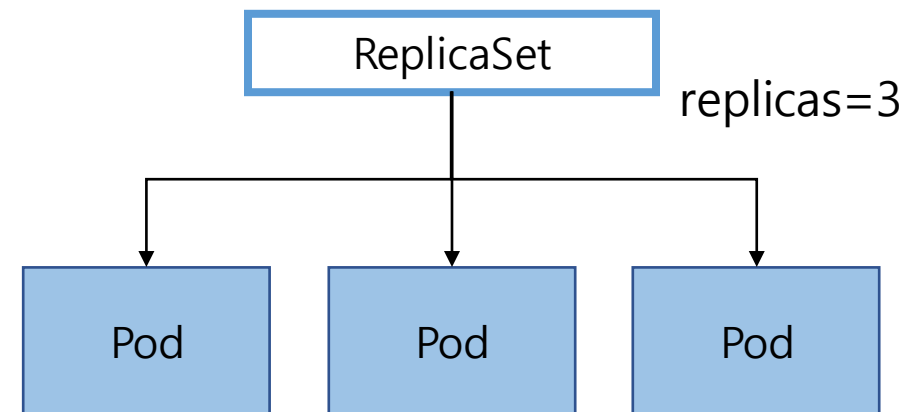
- 동일한 물리 클러스터에서 가상 클러스터를 나눠 지원하는 오브젝트
- 다른 네임스페이스 상에서는 같은 이름의 오브젝트가 존재 가능



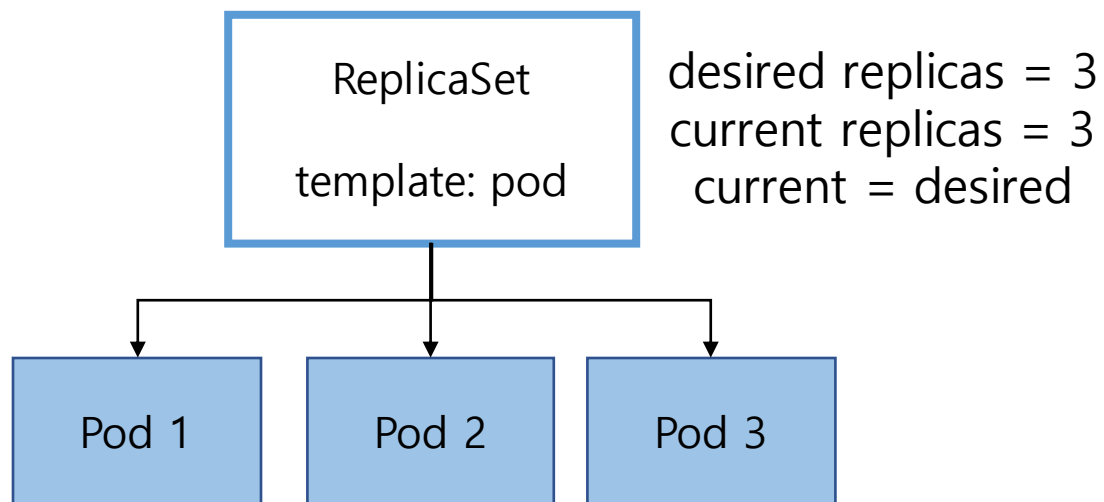
Kubernetes Controller - ReplicaSet

- ReplicaSet은 Pod의 개수를 유지
- yaml 을 작성할 때 replica 개수를 지정하면 그 개수에 따라 유지

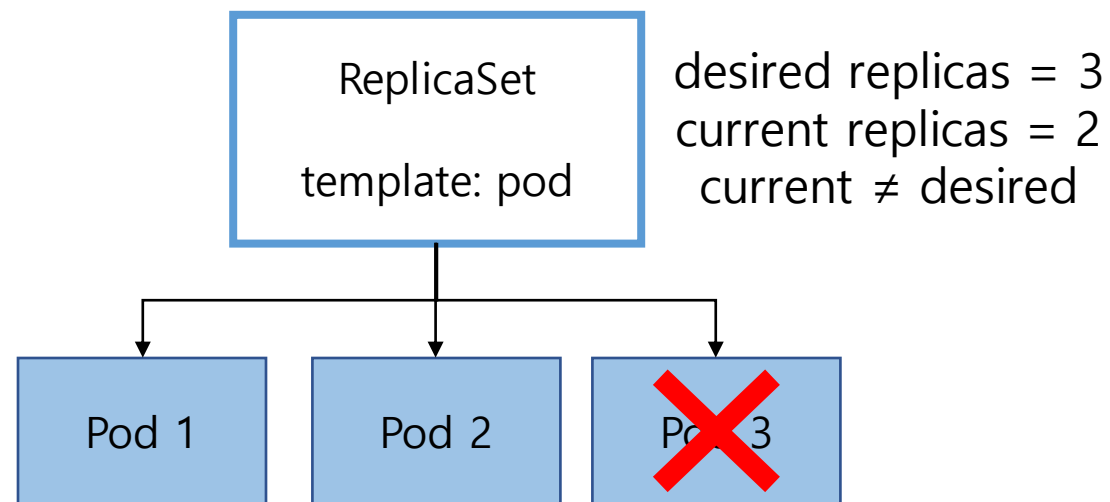
```
apiVersion: v1
kind: ReplicaSet
metadata:
  name: rep1
spec:
  replicas: 3
```



ReplicaSet Example -1

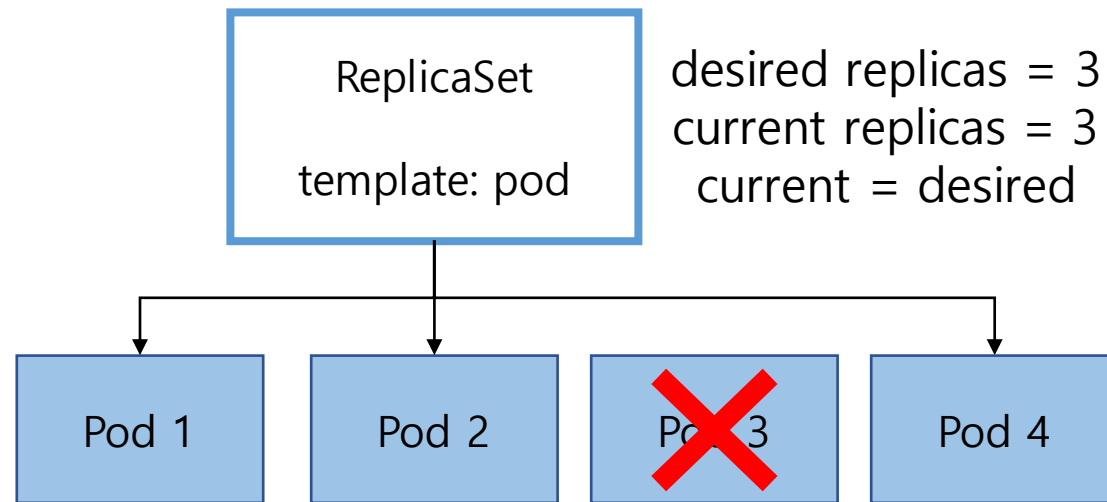


현재상태와 정의된 상태가 동일한 상황



현재상태와 정의된 상태가 달라진 상황

ReplicaSet Example -2



Pod 4가 새로 생성되며 현재상태와 정의된 상태가 동일

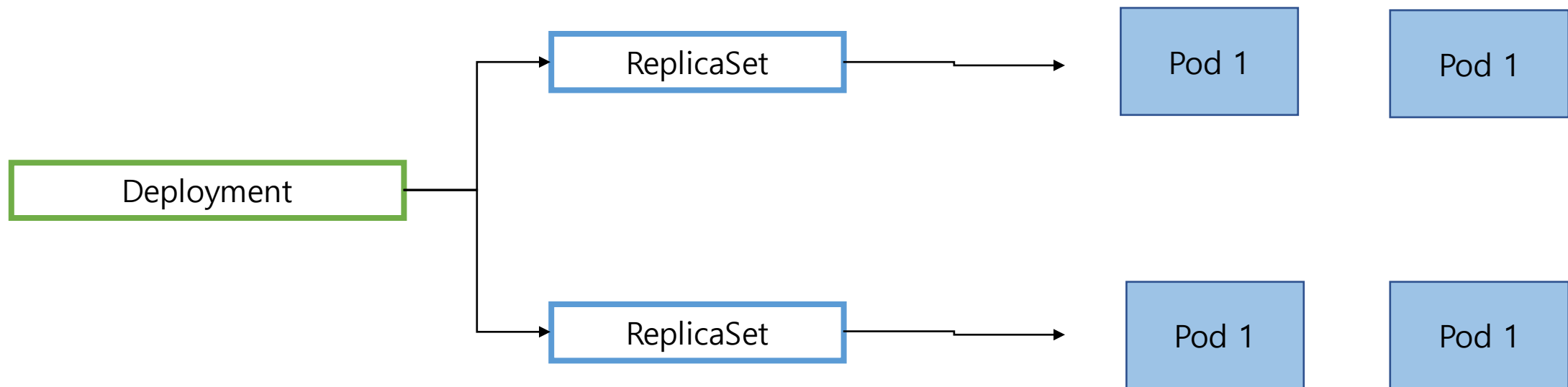
Template

- ㅣ 파드를 생성하기 위한 명세
- ㅣ Deployment, ReplicaSet과 같은 Controller의 yaml 내용에 포함
- ㅣ Template에는 Pod 세부사항을 결정

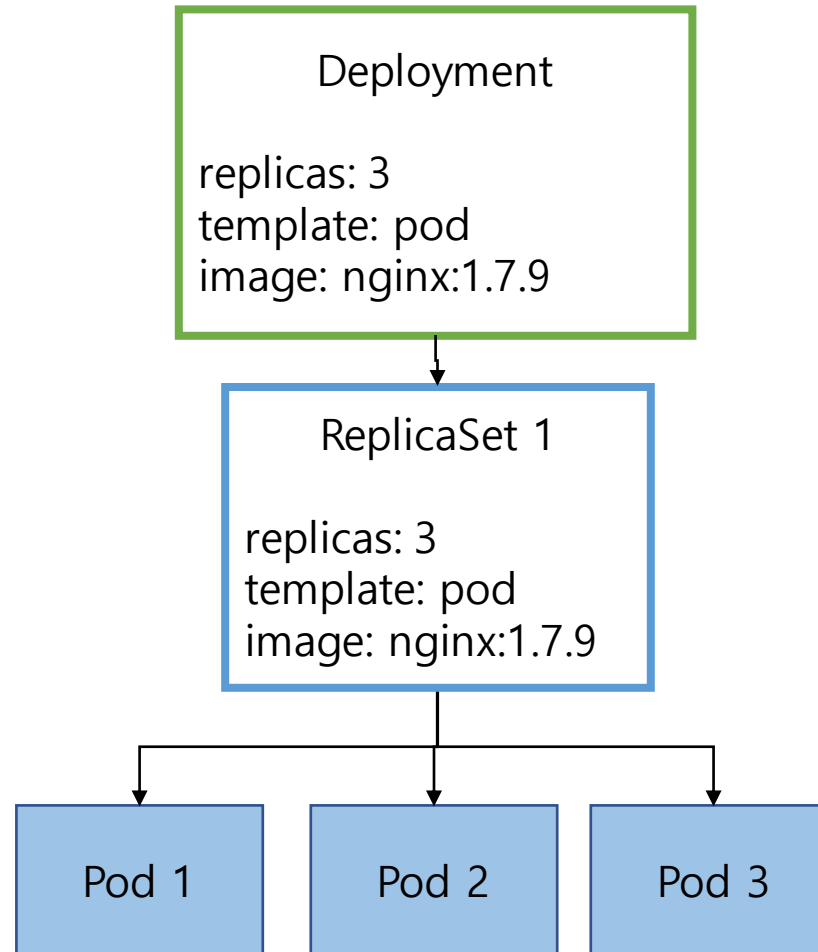
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
... 종략 ...
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
```

Kubernetes Controller – Deployment

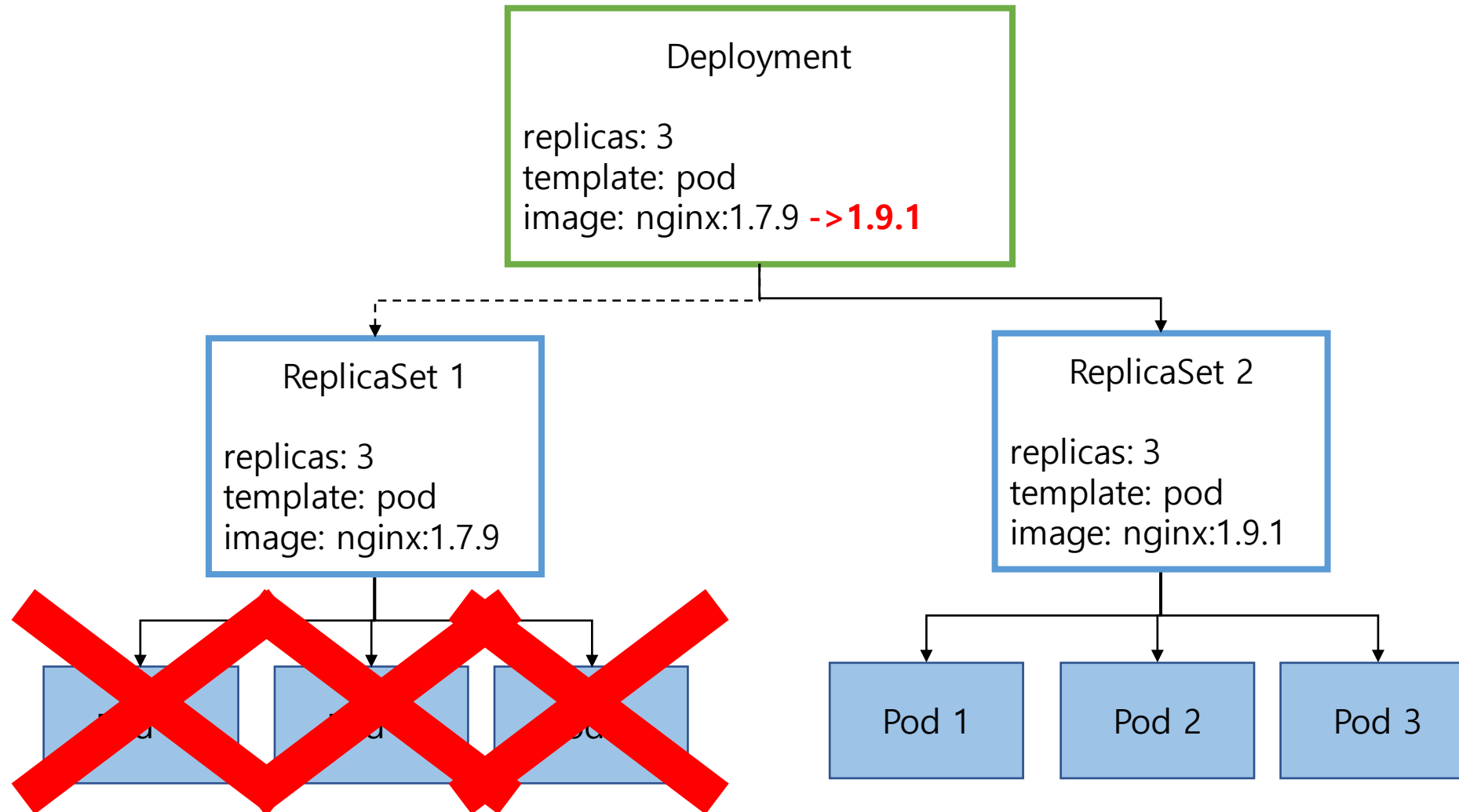
- Deployment는 ReplicaSet을 관리하며 애플리케이션의 배포를 더욱 세밀하게 관리
- 초기 배포 이후에 버전 업데이트, 이전 버전으로 Rollback도 가능



Deployment Example



Deployment Example



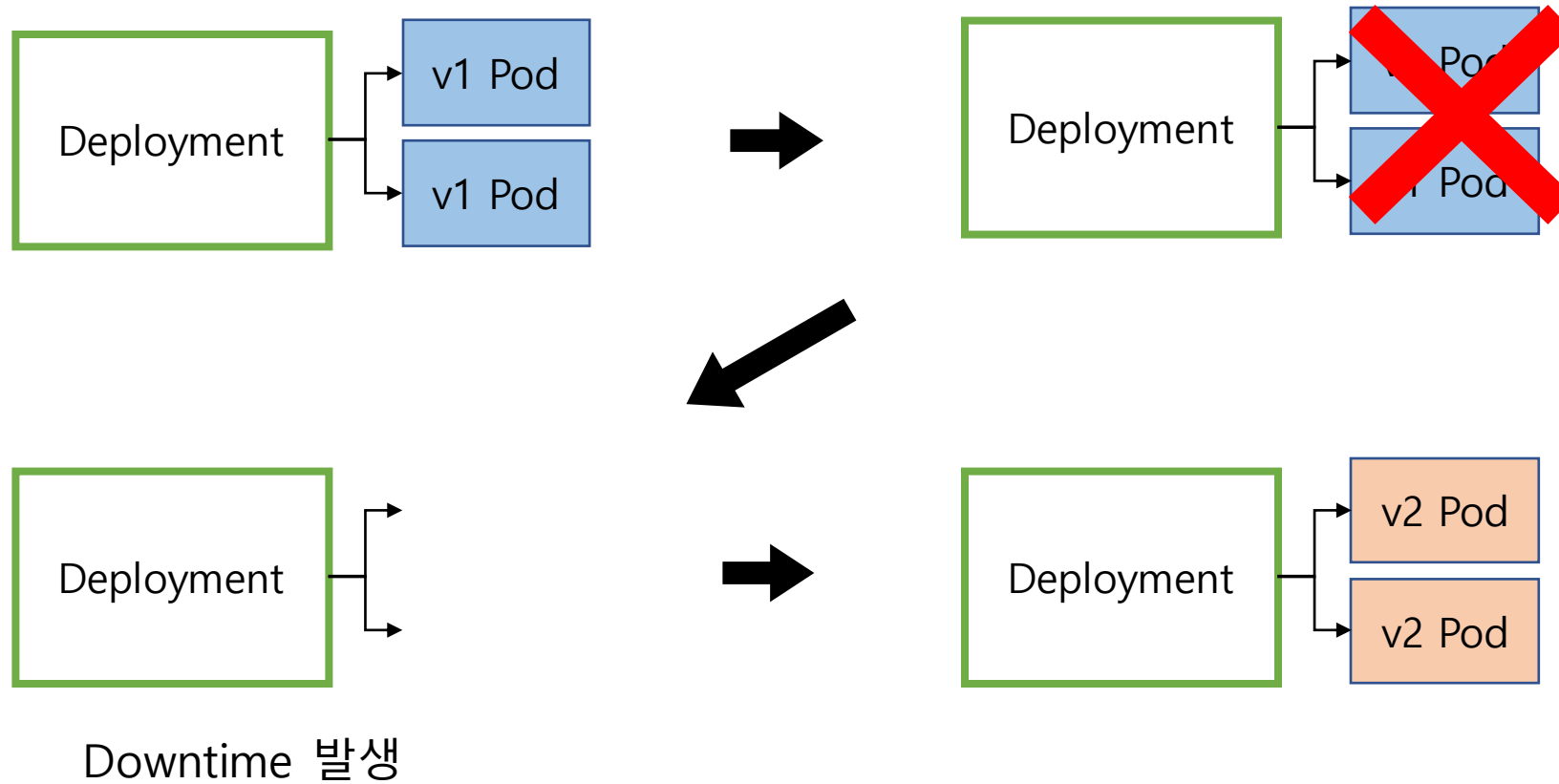
Deployment Update

I 운영중인 서비스의 업데이트시 재배포를 관리

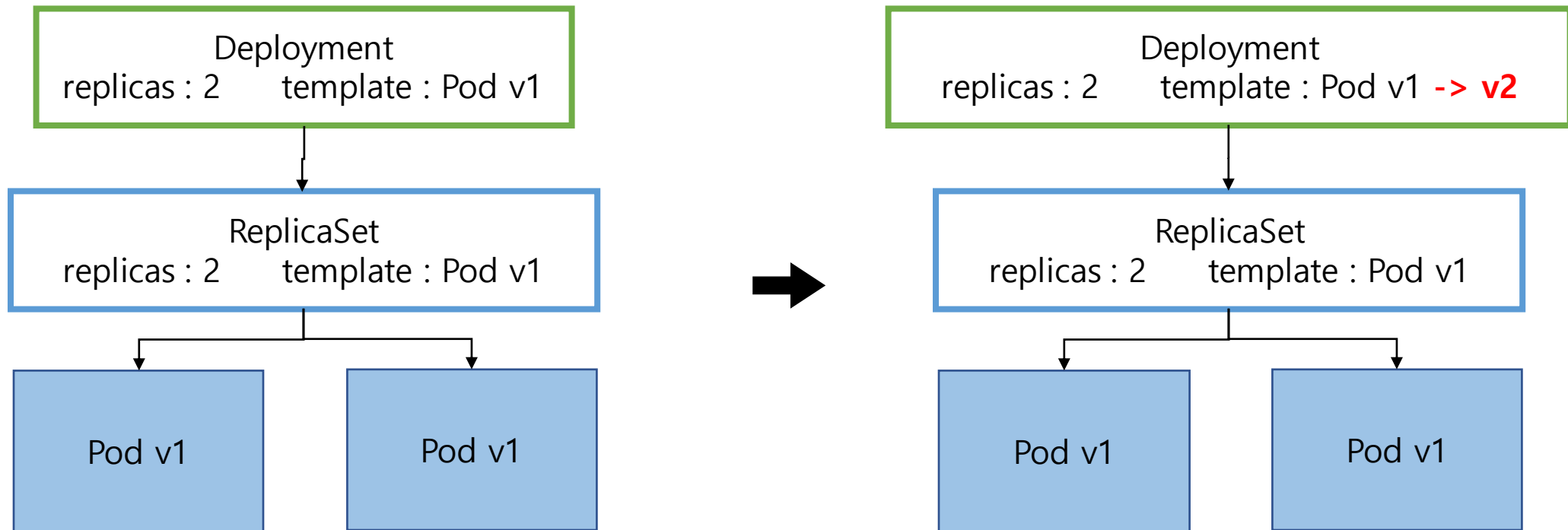
I 2가지 재배포 방식

- ▶ Recreate : 현재 운영중인 Pod들을 삭제하고, 업데이트 된 Pod들을 생성합니다. 이 방식은 Downtime 이 발생하기 때문에 실시간으로 사용해야 한다면 권장하는 방식은 아닙니다.
- ▶ Rolling Update : 먼저 업데이트 된 Pod를 하나 생성하고, 구버전의 Pod를 삭제하여, Downtime 없이 업데이트가 가능합니다.

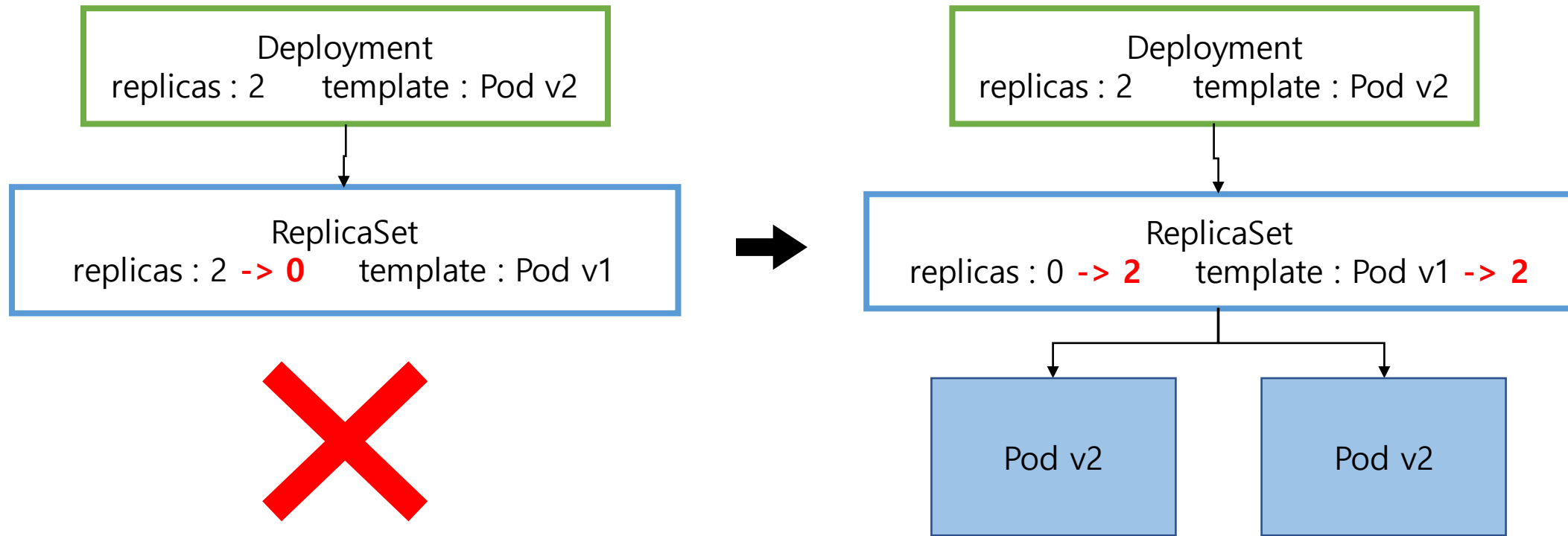
Recreate -1



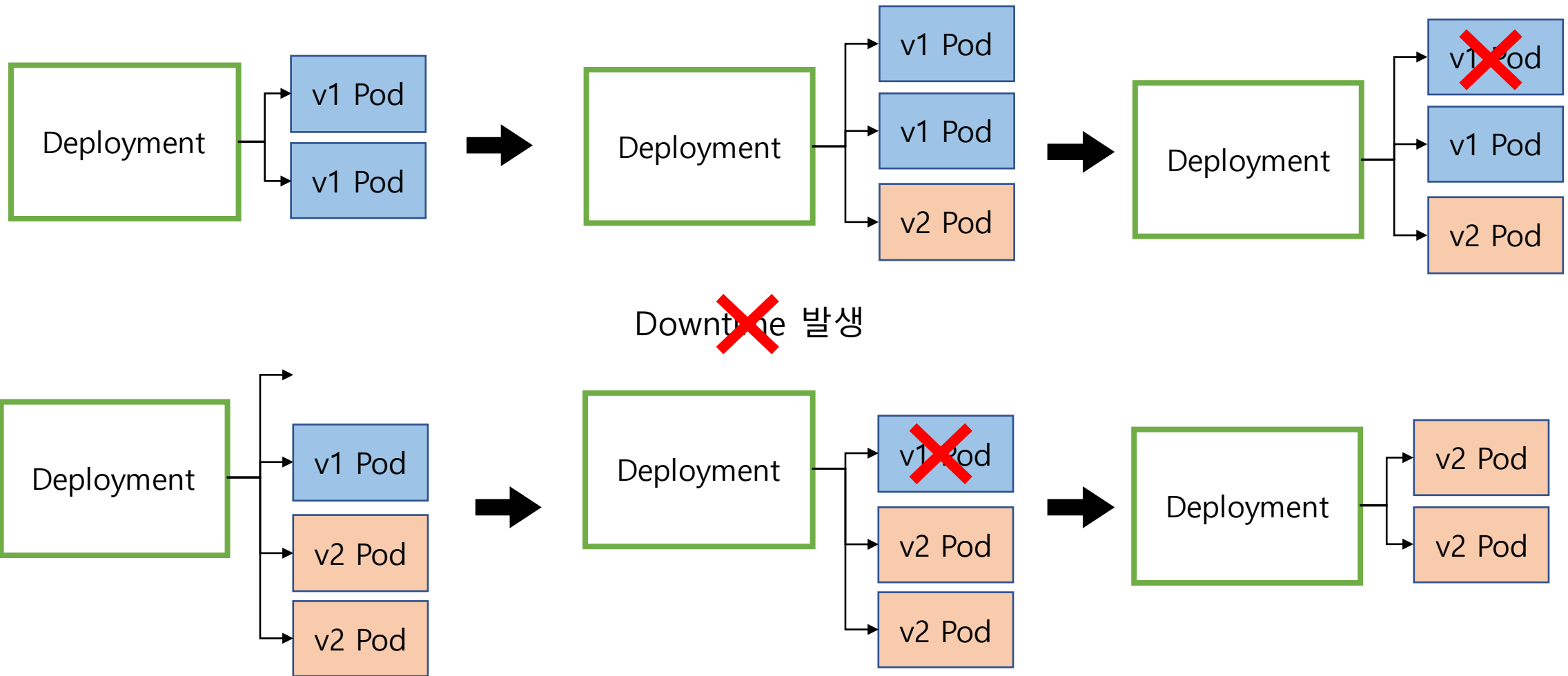
Recreate -2



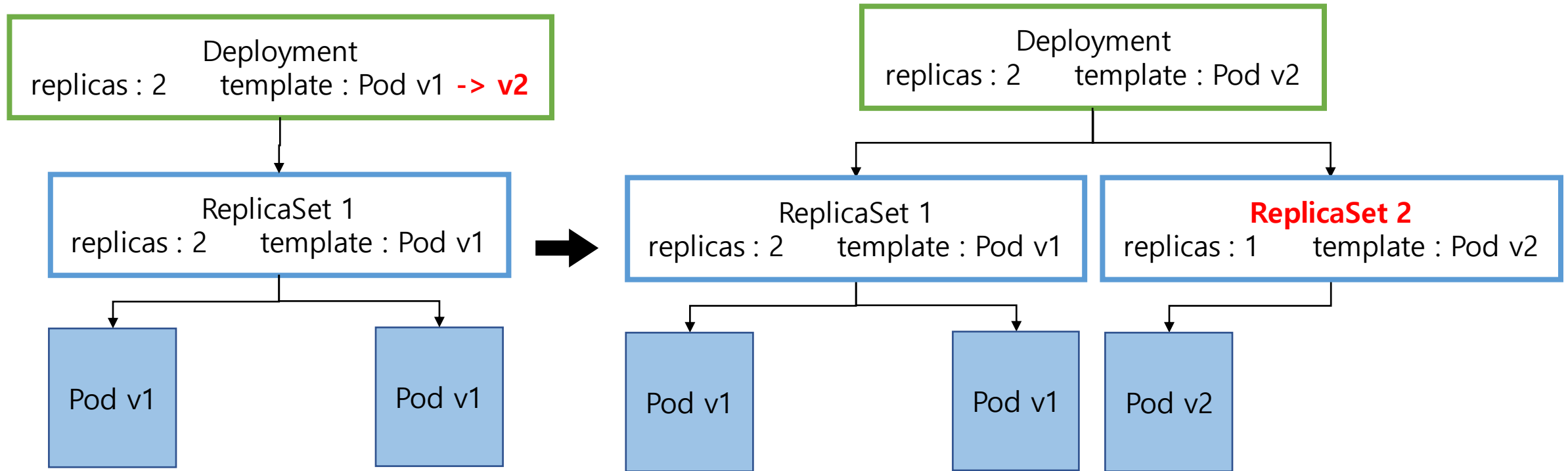
Recreate -3



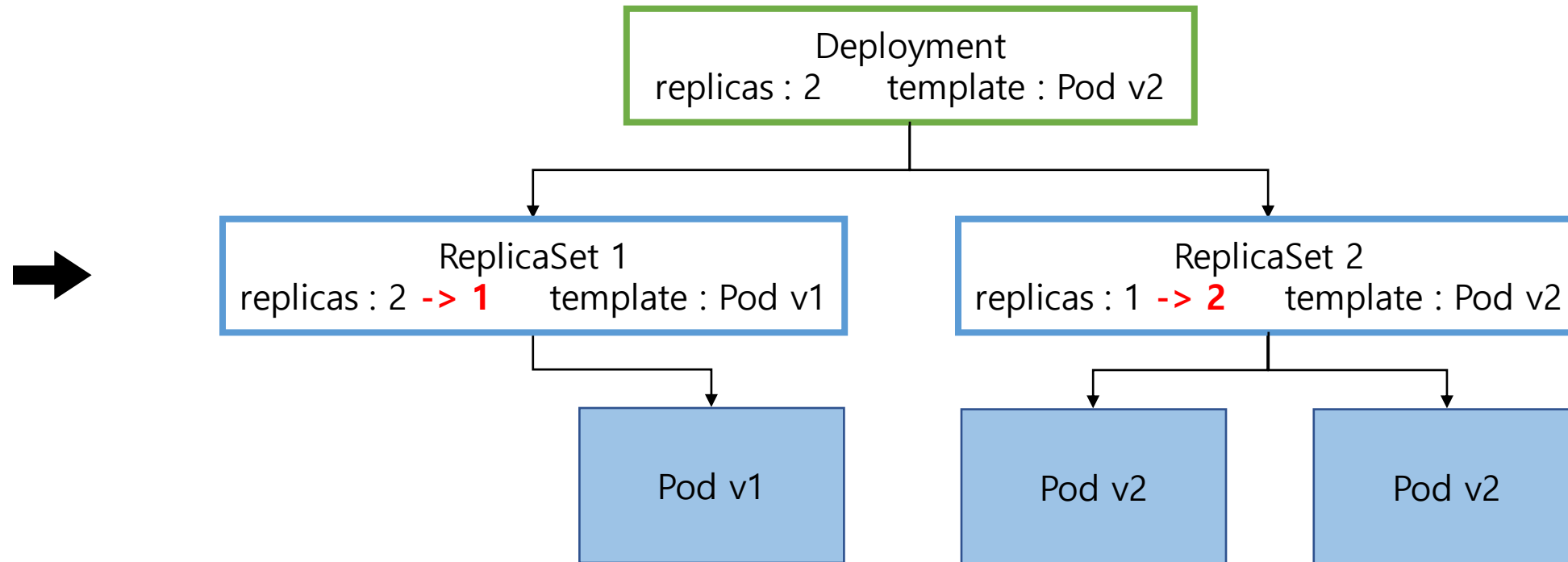
Rolling Update -1



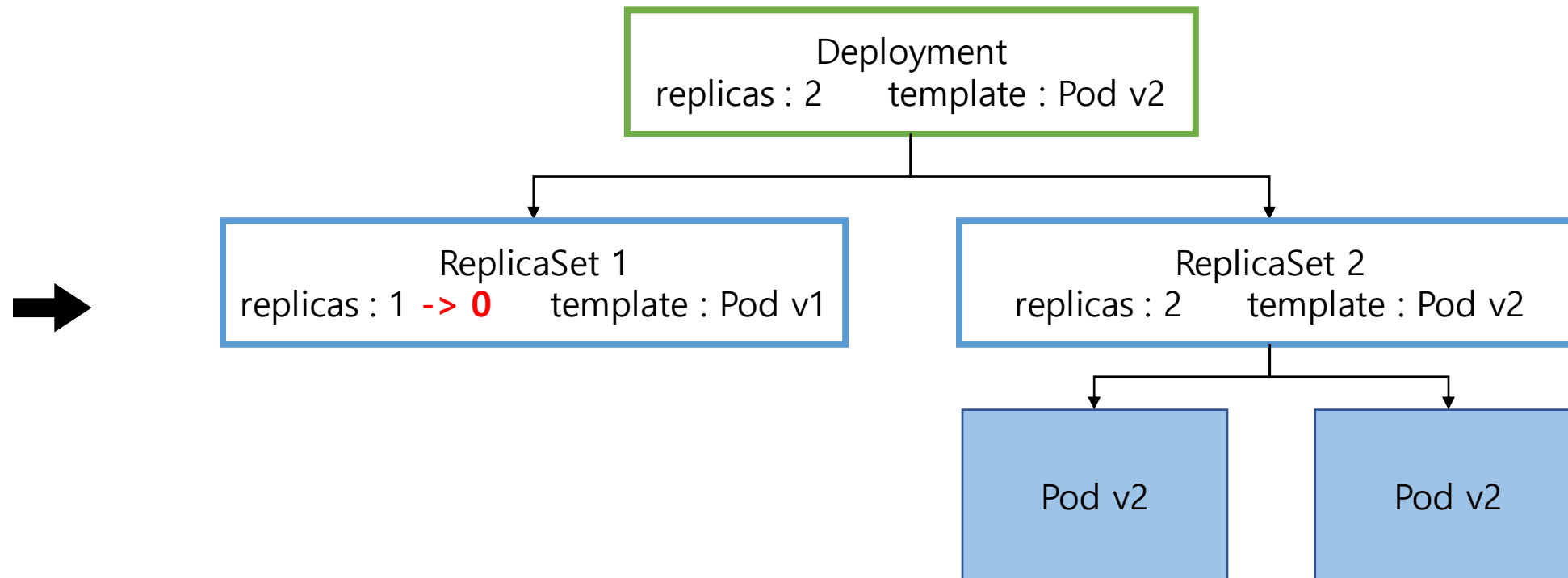
Rolling Update -2



Rolling Update -3



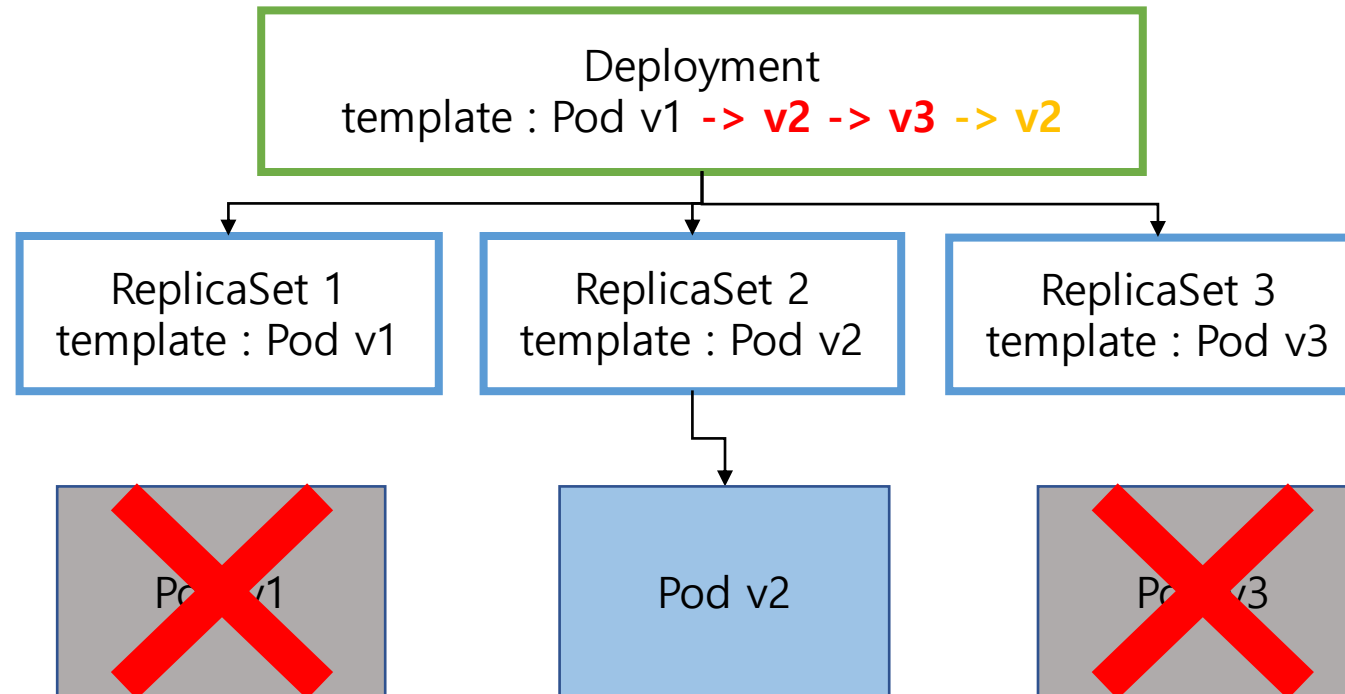
Rolling Update -4



Deployment Rollback

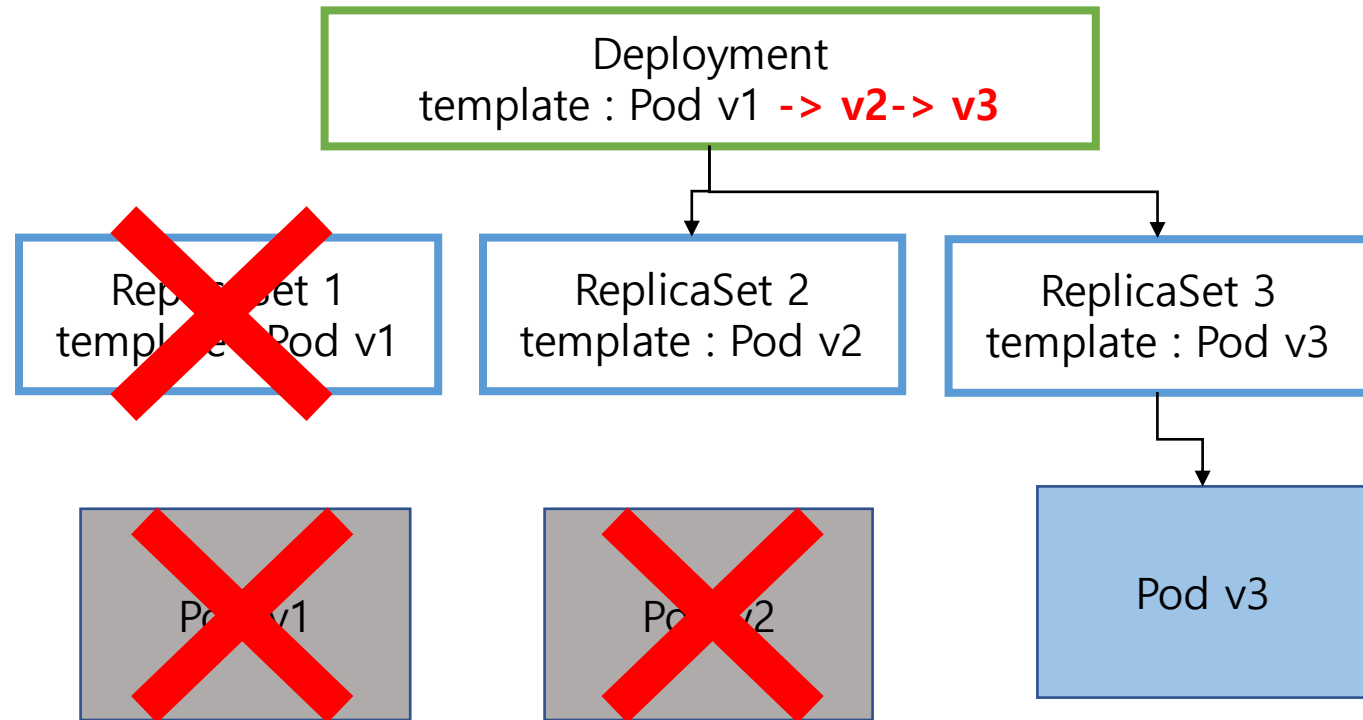
- Deployment는 이전버전의 ReplicaSet을 2개까지 저장
- 저장된 이전 버전의 ReplicaSet을 활용하여 Rollback 가능
- revisionHistoryLimit 속성을 설정하면 저장할 이전버전의 ReplicaSet의 개수를 변경 가능

Rollback -1



Rollback -2

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment1
spec:
  replicas: 1
  strategy:
    type: Recreate
  revisionHistoryLimit: 1
```



Deployment - Kubectl

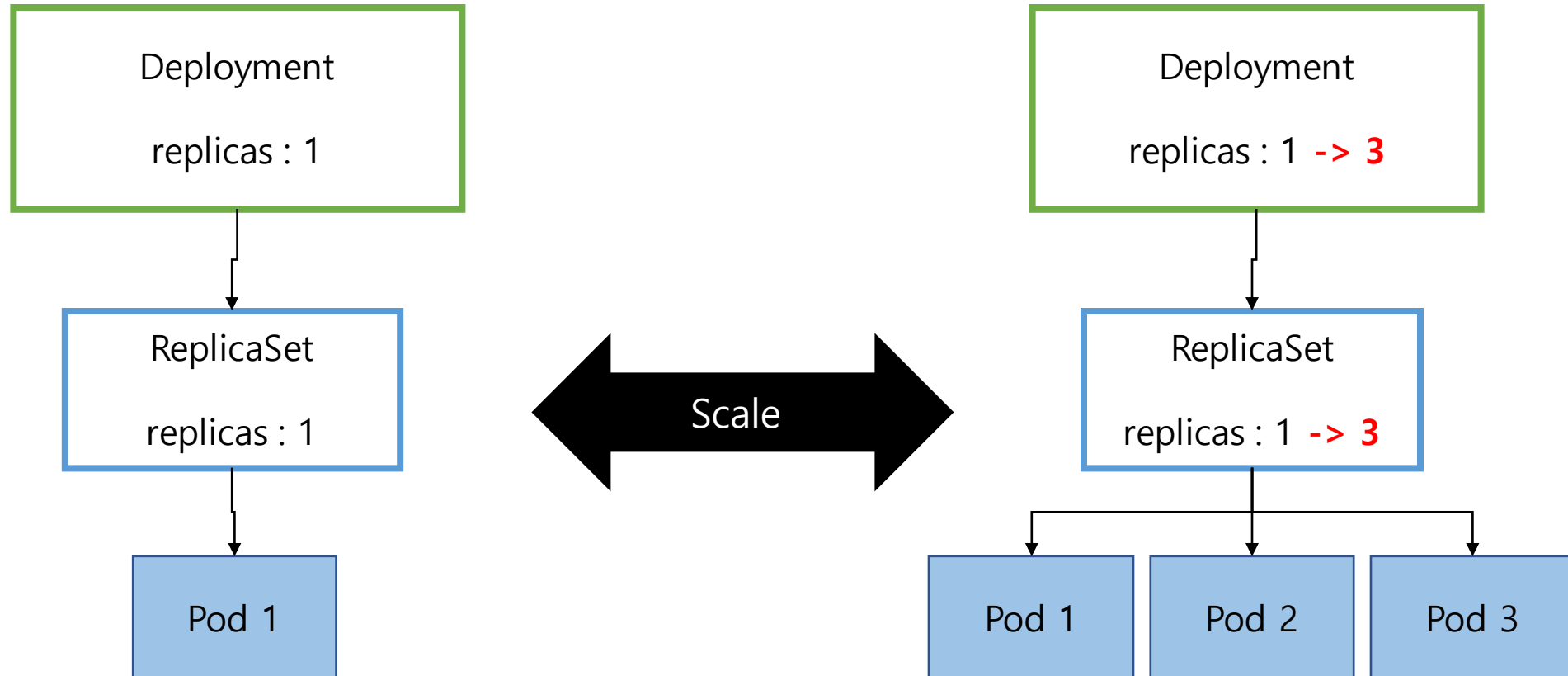
I yaml 파일을 사용하여 생성 하는 명령어

- ▶ `kubectl create -f deployment.yaml`
- ▶ `kubectl create -f <yaml 파일명>`

I kubectl 명령으로 생성하는 명령어

- ▶ `kubectl create deployment dp --image=nginx:1.14.0 --replicas=3`
- ▶ `kubectl create deployment <이름> --image=<이미지명:버전> --replicas=<Pod수>`

Scale



Scale - kubectl

I Deployment로 생성된 Pod 수를 조정하는 명령어

- ▶ `kubectl scale deployment/dp --replicas=3`
- ▶ `kubectl scale deployment/<Deployment명> --replicas=<조정할 Pod 수>`

I ReplicaSet으로 생성된 Pod 수를 조정하는 명령어

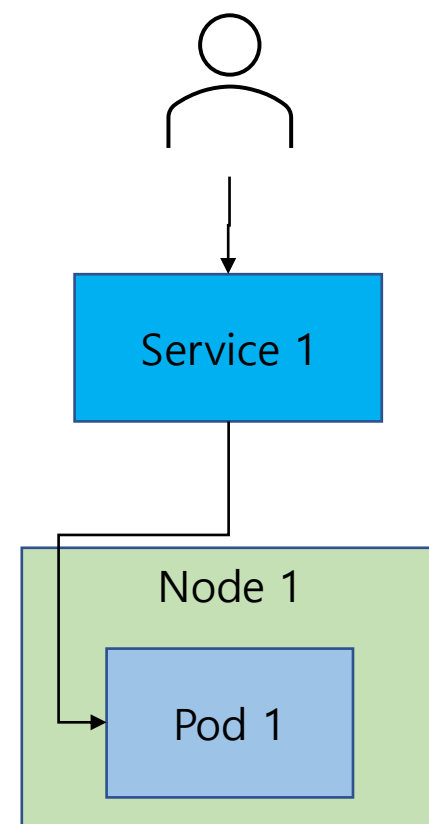
- ▶ `kubectl scale rs/rs --replicas=3`
- ▶ `kubectl scale rs/<ReplicaSet명> --replicas=<조정할 Pod 수>`

Chapter 4

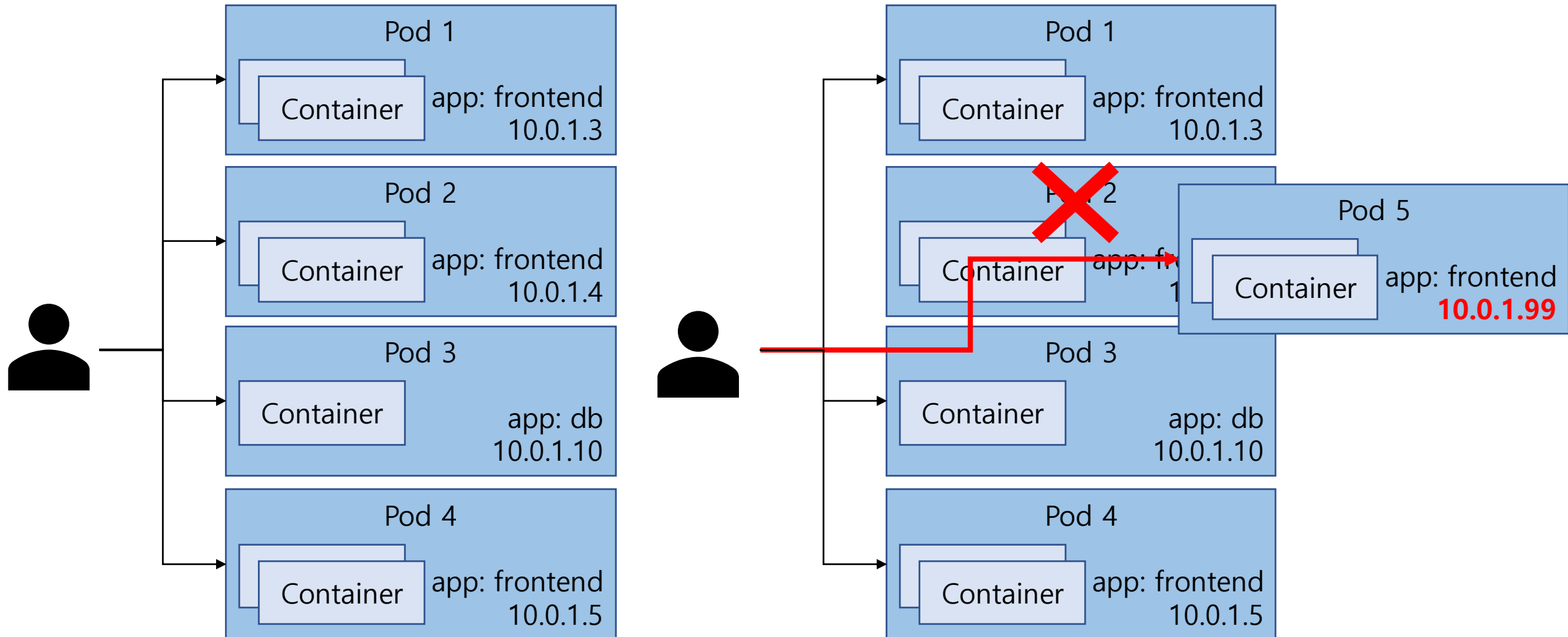
I 컨테이너 통신 - Service, Ingress

Kubernetes Object - Service

- Pod에 접근하기 위해 사용하는 Object
- Kubernetes 외부 또는 내부에서 Pod에 접근할 때 필요
- 고정된 주소를 이용하여 접근이 가능
- Pod에서 실행중인 애플리케이션을 네트워크 서비스로 노출시키는 Object



Service Motivation



Label

- Pod와 같은 Object에 첨부된 키와 값 쌍

```
apiVersion: v1
kind: Pod
metadata:
  name: label
  labels:
    environment: production
    app: nginx
```

Label Selector

- 특정 Label 값을 찾아 해당하는 Object만 관리할 수 있게 연결

```
apiVersion: v1
kind: Service
metadata:
  name: service1
spec:
  selector:
    app: nginx
```

Annotation

- Object를 식별하고 선택하는 데에는 사용되지 않으나 참조할 만한 내용들을 Label처럼 첨부

```
apiVersion: v1
kind: Pod
metadata:
  name: annotations-demo
  annotations:
    imageregistry: "https://hub.docker.com/"
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```


Service 유형

I ClusterIP(default)

- ▶ Service가 기본적으로 갖고있는 ClusterIP를 활용하는 방식입니다.

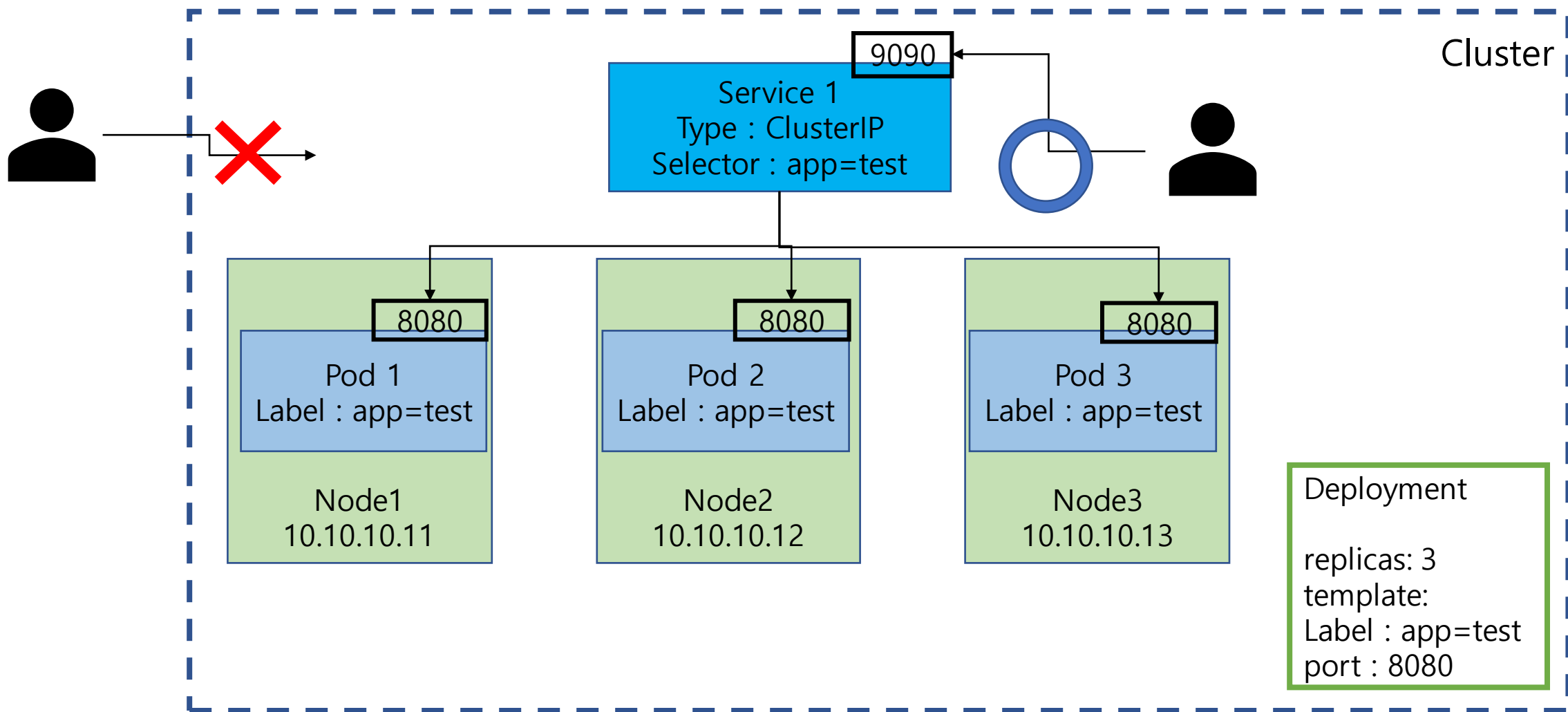
I NodePort

- ▶ 모든 Node에 Port를 할당하여 접근하는 방식입니다.

I Load Balancer

- ▶ Load Balancer Plugin 을 설치하여 접근하는 방식입니다.

ClusterIP



ClusterIP YAML

```
apiVersion: v1
kind: Service
metadata:
  name: svc1
spec:
  selector:
    app: pod
  ports:
    - port: 9090
      targetPort: 8080
  type: ClusterIP
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    app: pod
spec:
  containers:
    - name: container
      image: cluster/app
      ports:
        - containerPort: 8080
```

ClusterIP - kubectl

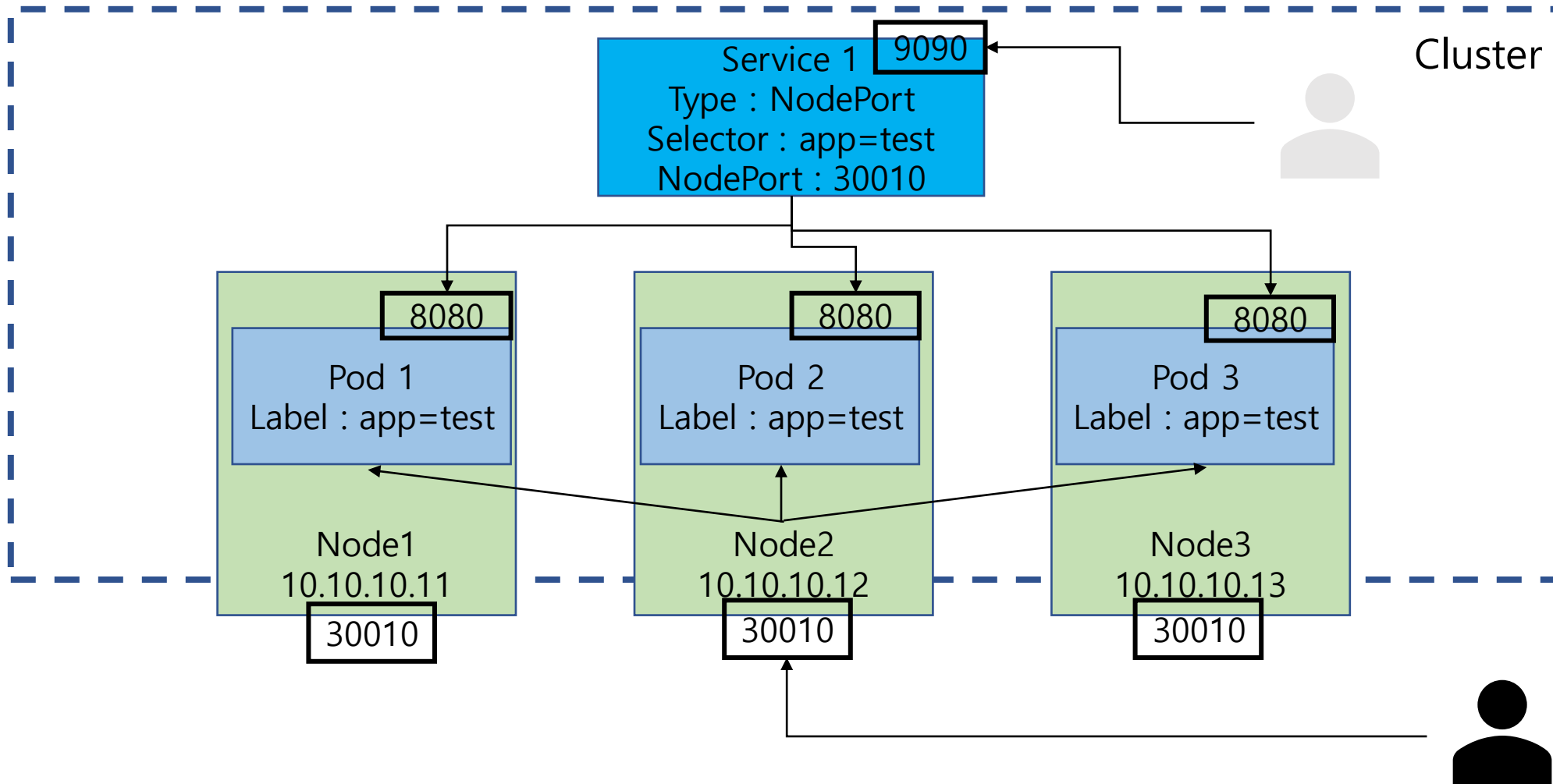
I ClusterIP 유형의 Service를 생성하는 명령어

- ▶ `kubectl create service clusterip clip --tcp=5678:8080`
- ▶ `kubectl create service clusterip <Service명> --tcp=<포트:타켓포트>`

I ClusterIP 유형의 Service를 nginx라는 Deployment와 연결하여 생성하는 명령어

- ▶ `kubectl expose deployment nginx --port=80 --target-port=8000 --type=ClusterIP`
- ▶ `kubectl expose <연결할오브젝트> <오브젝트명> --port=<포트> --target-port=<타
겟포트> --type=ClusterIP`

NodePort



NodePort YAML

- 해당 Node의 IP와 Node에 할당된 포트 번호 30010으로 접속을 하면 해당 서비스에 연결
- Service는 자신에게 연결 되어있는 Pod에 트래픽을 전달

```
apiVersion: v1
kind: Service
metadata:
  name: svc2
spec:
  selector:
    app: pod
  ports:
    - port: 9090
      targetPort: 8080
      nodePort: 30010
  type: NodePort
  externalTrafficPolicy: Local
```

NodePort - kubectl

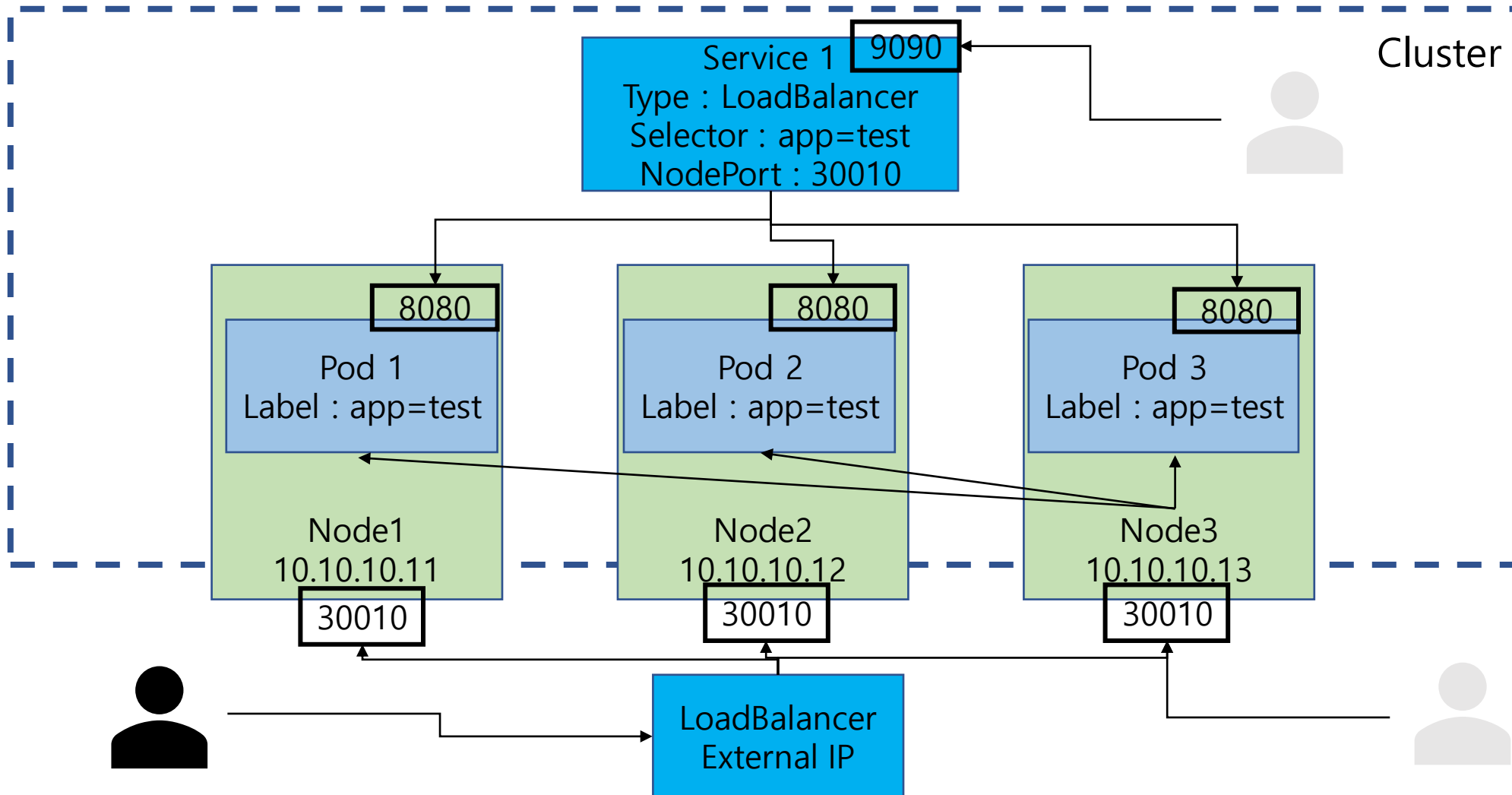
I NodePort 유형의 Service를 생성하는 명령어

- ▶ `kubectl create service nodeport np --tcp=5678:8080`
- ▶ `kubectl create service nodeport <Service명> --tcp=<포트:타겟포트>`

I NodePort 유형의 Service를 nginx라는 Deployment와 연결하여 생성하는 명령어

- ▶ `kubectl expose deployment nginx --port=80 --target-port=8000 --type=NodePort`
- ▶ `kubectl expose <연결할오브젝트> <오브젝트명> --port=<포트> --target-port=<타겟포트> --type=NodePort`

Load Balancer



Load Balancer YAML

- 온전한 Load Balancer 의 기능을
사용하려면, 추가 플러그인 설치가
필요
- 또는 로드밸런서를 지원하는 클
라우드 환경에서 사용가능

```
apiVersion: v1
kind: Service
metadata:
  name: svc3
spec:
  selector:
    app: pod
  ports:
    - port: 9090
      targetPort: 8080
  type: LoadBalancer
```

LoadBalancer - kubectl

I LoadBalancer 유형의 Service를 생성하는 명령어

- ▶ `kubectl create service loadbalancer lb --tcp=5678:8080`
- ▶ `kubectl create service loadbalancer <Service명> --tcp=<포트:타겟포트>`

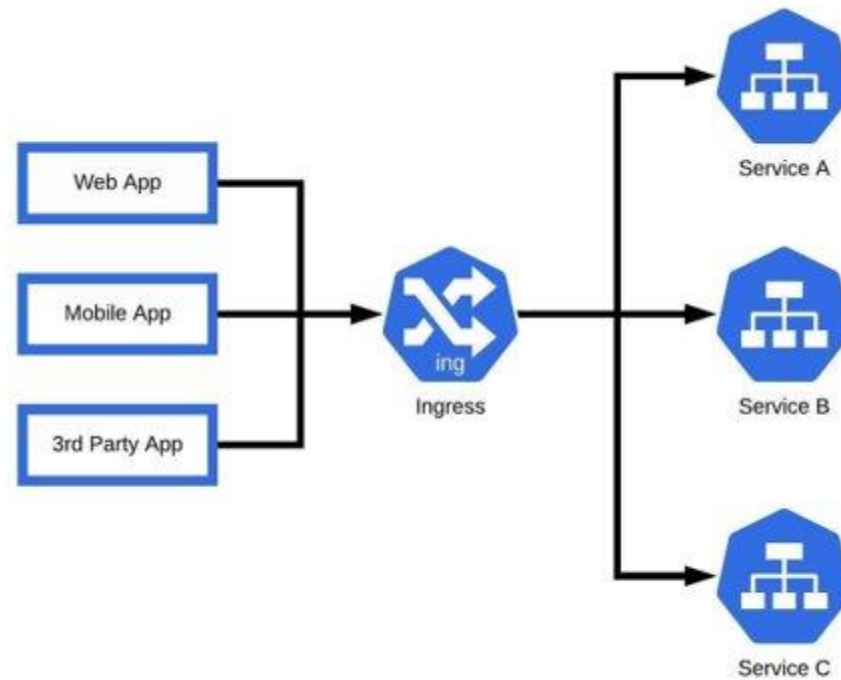
I Loadbalancer 유형의 Service를 nginx라는 Deployment와 연결하여 생성하는 명령어

- ▶ `kubectl expose deployment nginx --port=80 --target-port=8000 --type=LoadBalancer`
- ▶ `kubectl expose <연결할오브젝트> <오브젝트명> --port=<포트> --target-port=<타겟포트> --type=NodePort`

Ingress

- | 클러스터 내의 서비스에 대한 외부 접근을 관리하는 오브젝트
- | 일반적으로 HTTP를 관리
- | 부하분산, SSL 인증서 처리, Service에 외부 URL 제공

Ingress

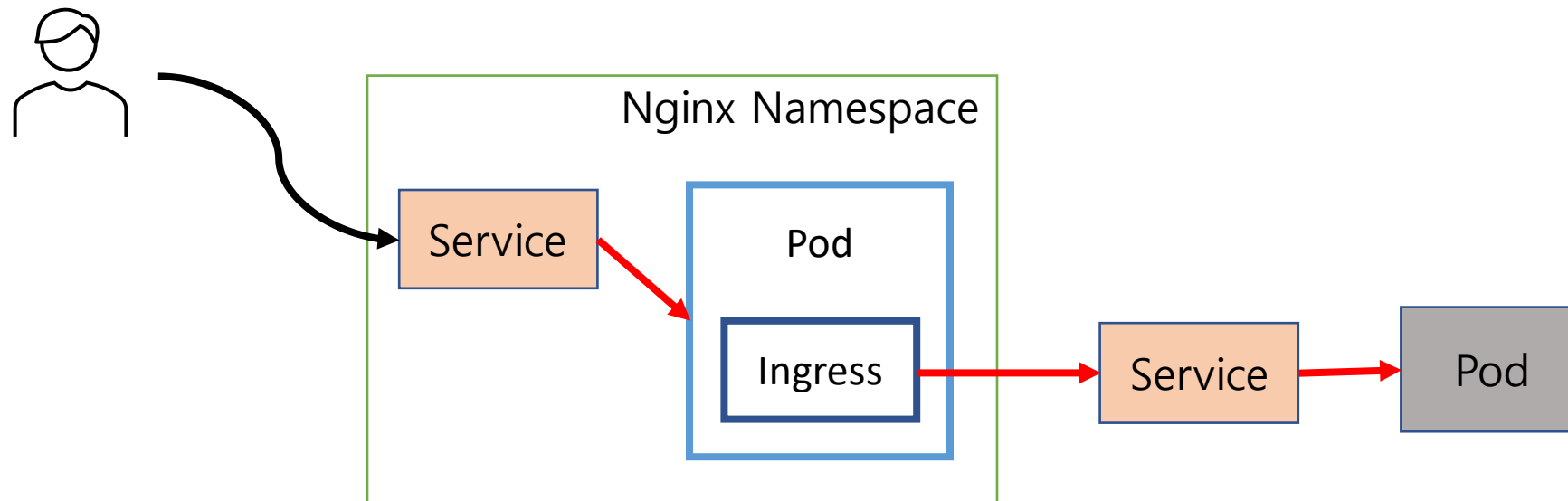


Ingress controller

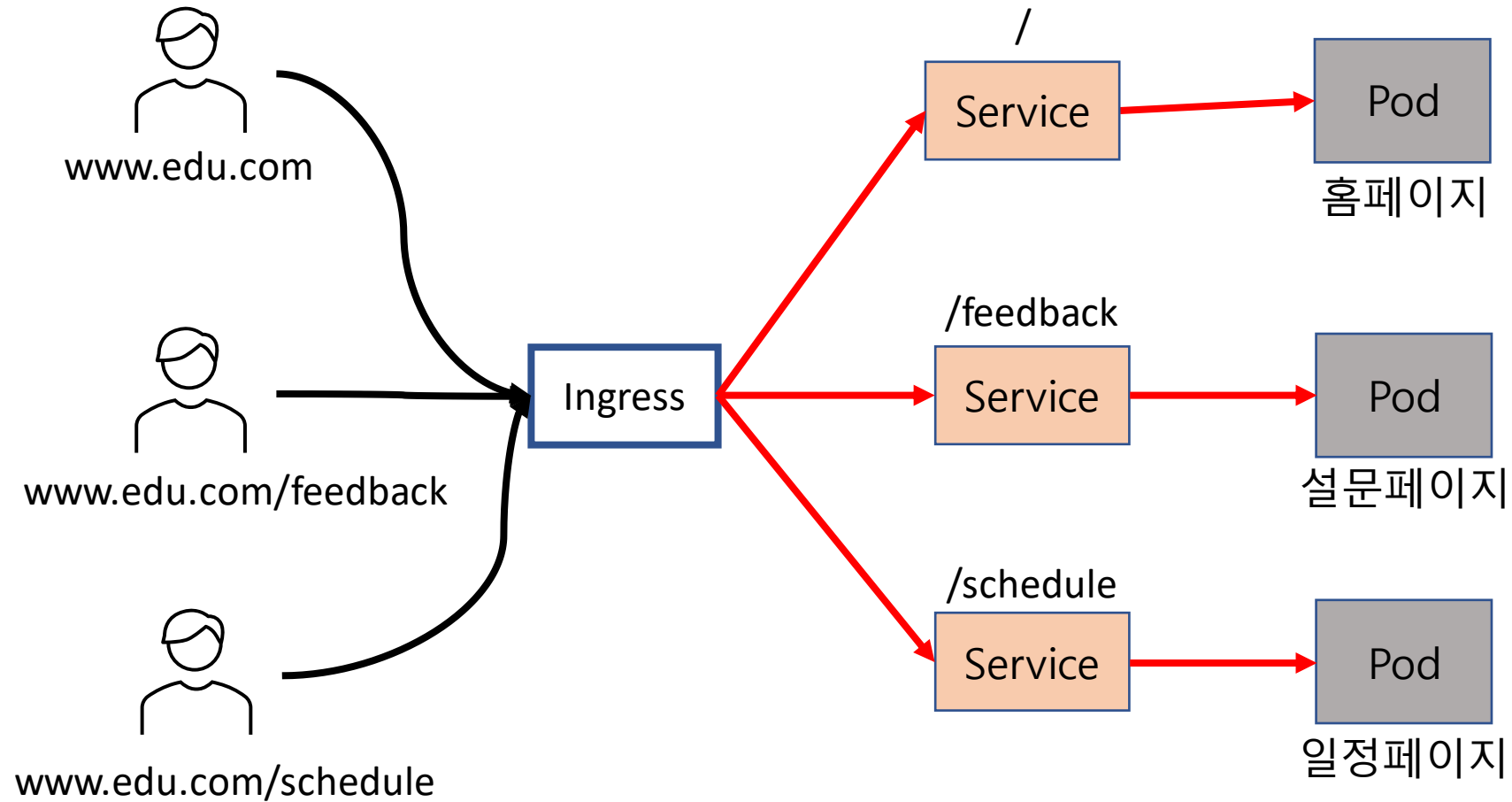
- | Kubernetes에서 공식적으로 제공하는 Ingress controller
 - ▶ AWS, GCE, nginx
- | 대표적으로는 nginx, Kong을 많이 사용



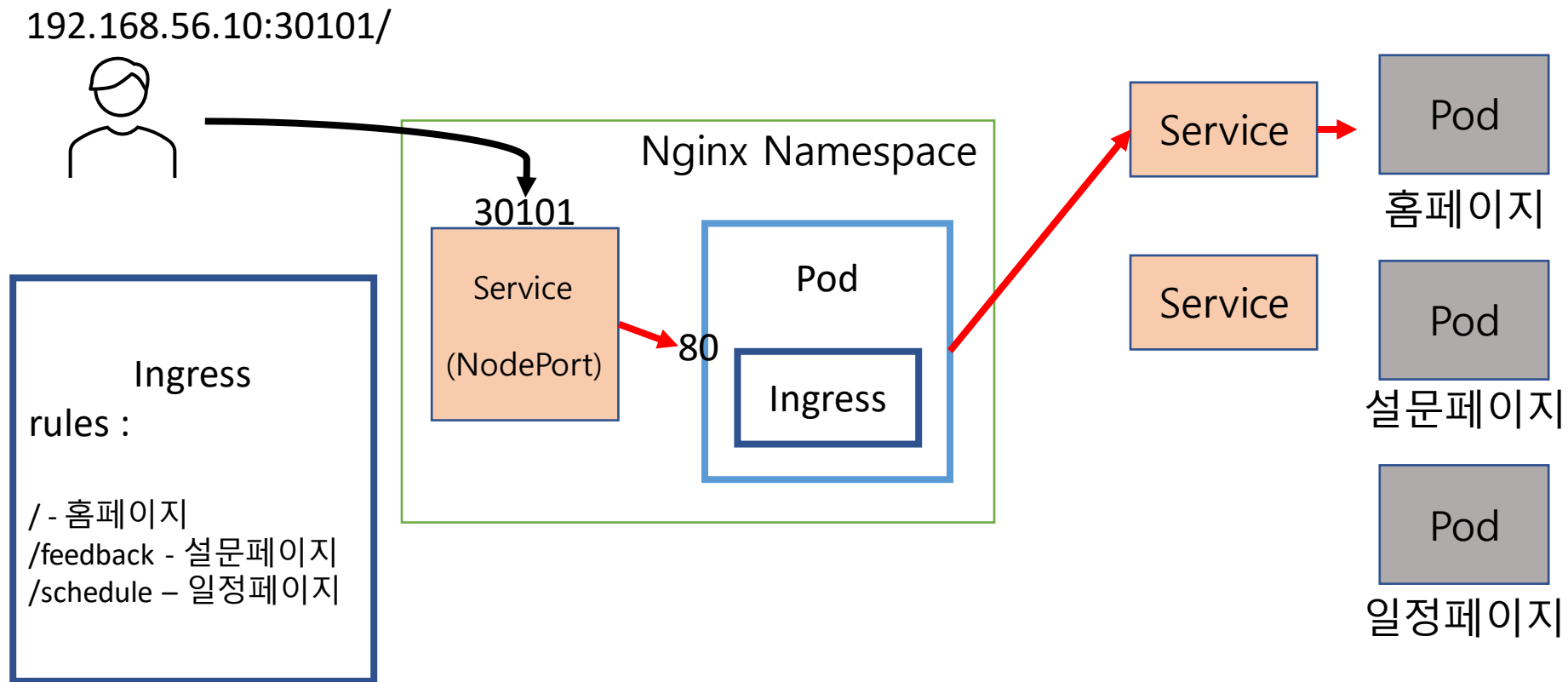
Ingress controller 구조



서비스 로드밸런싱



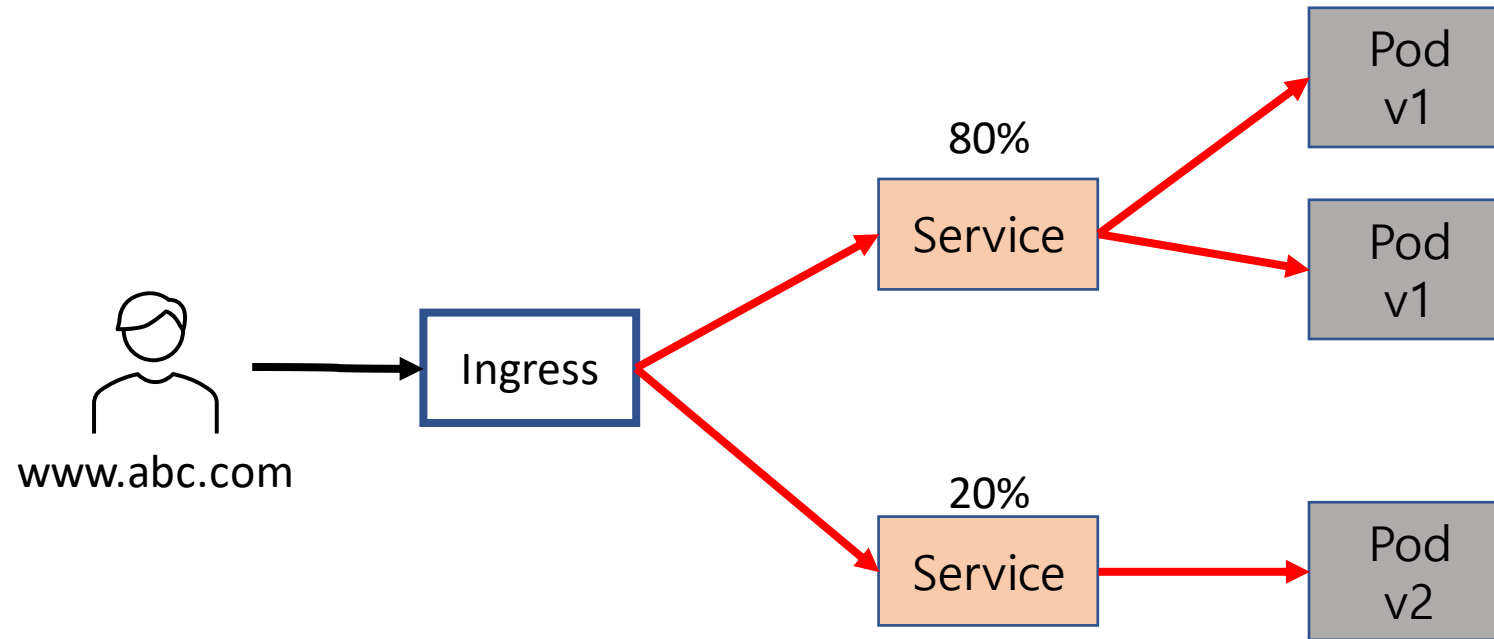
서비스 로드밸런싱



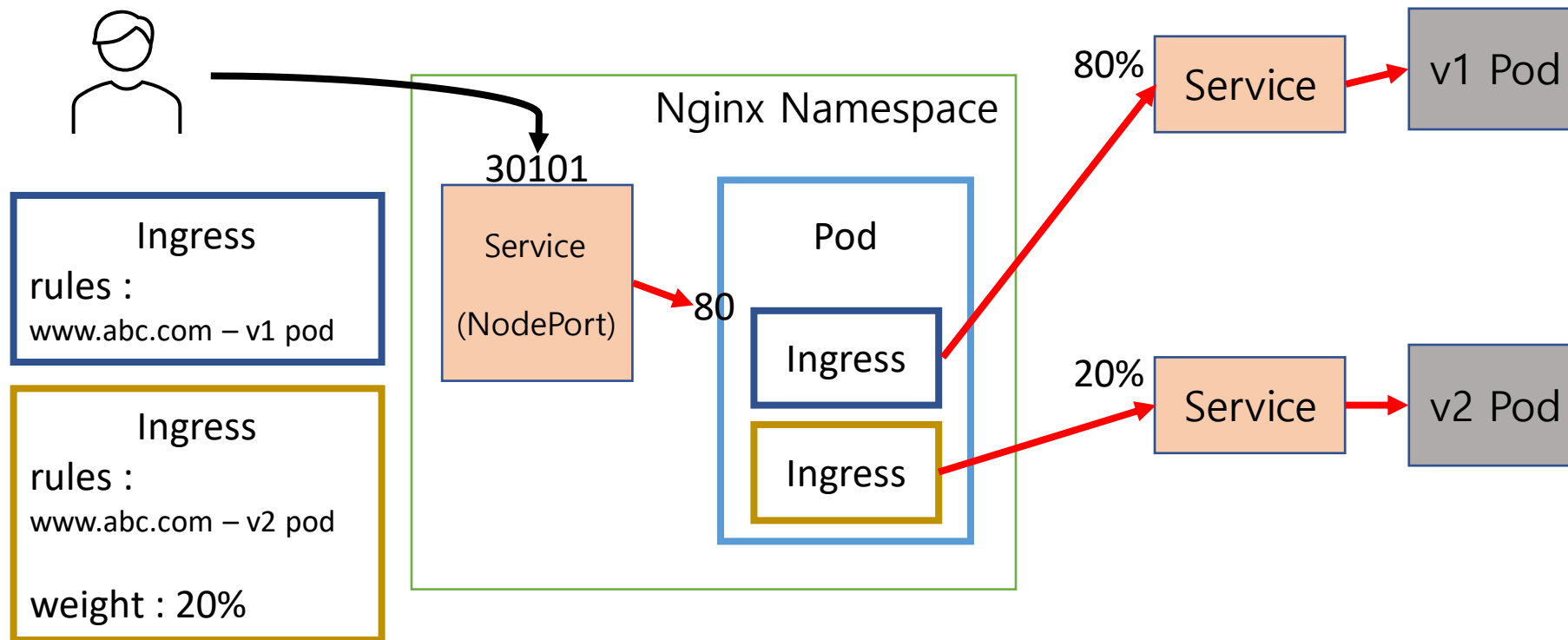
서비스 로드밸런싱 YAML

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: loadbalancing
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: homepage
          servicePort: 80
      - path: /feedback
        backend:
          serviceName: feedback
          servicePort: 80
      - path: /schedule
        backend:
          serviceName: schedule
          servicePort: 80
```

카나리 업그레이드



카나리 업그레이드



카나리 업그레이드 YAML

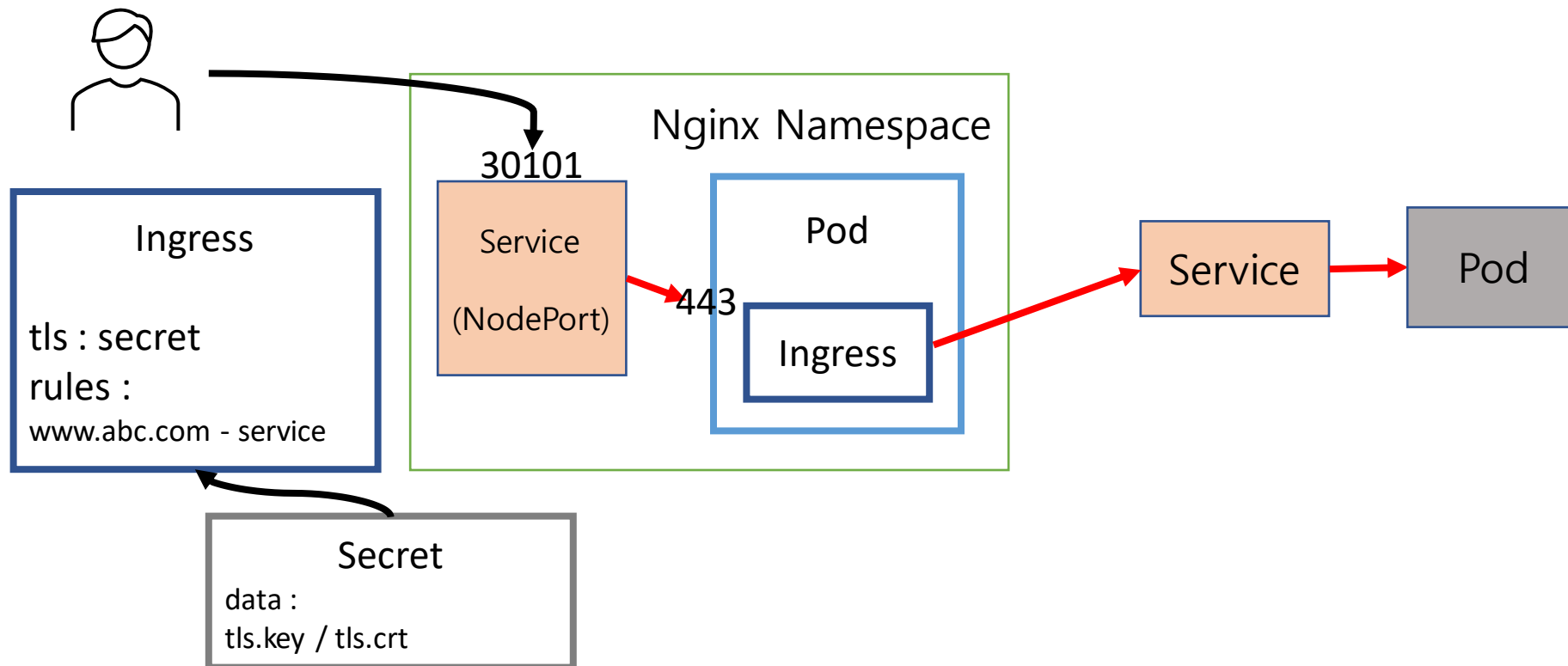
Ingress

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: v1
spec:
  rules:
  - host: www.abc.com
    http:
      paths:
      - backend:
          serviceName: svc1
          servicePort: 80
```

Ingress

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: v2
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "20"
spec:
  rules:
  - host: www.abc.com
    http:
      paths:
      - backend:
          serviceName: svc2
          servicePort: 80
```

https 인증서 관리



https 인증서 관리 YAML

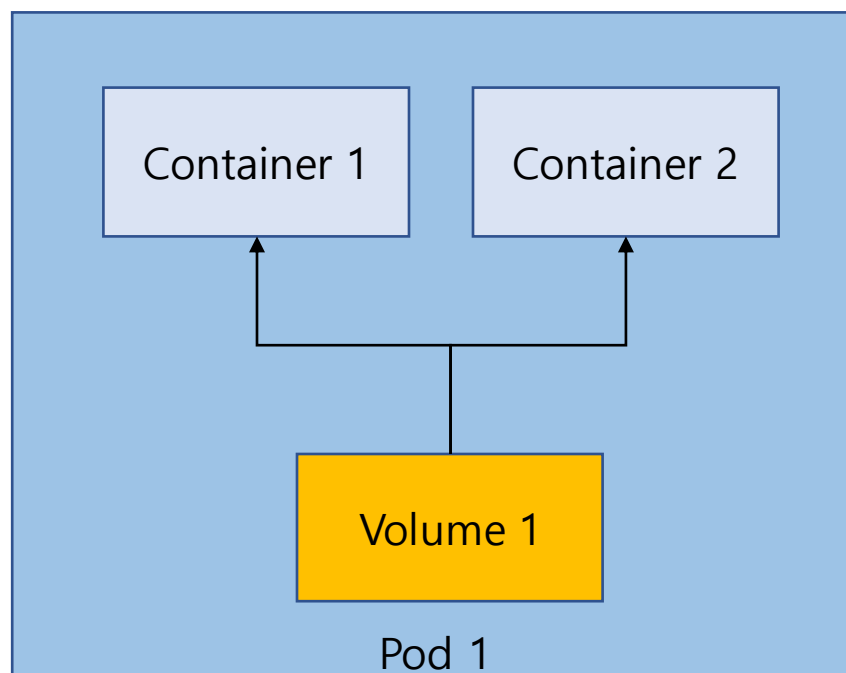
```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: https
spec:
  tls:
  - hosts:
    - www.abc.com
    secretName: secret
  rules:
  - host: www.abc.com
    http:
      paths:
      - backend:
          serviceName: svc
          servicePort: 80
```

Chapter 5

I 컨테이너 볼륨 및 환경변수 - Volume, Configmap, Secret

Volume

- Kubernetes에서 Volume은 Pod 컨테이너에서 접근할 수 있는 디렉터리라고 생각할 수 있습니다.



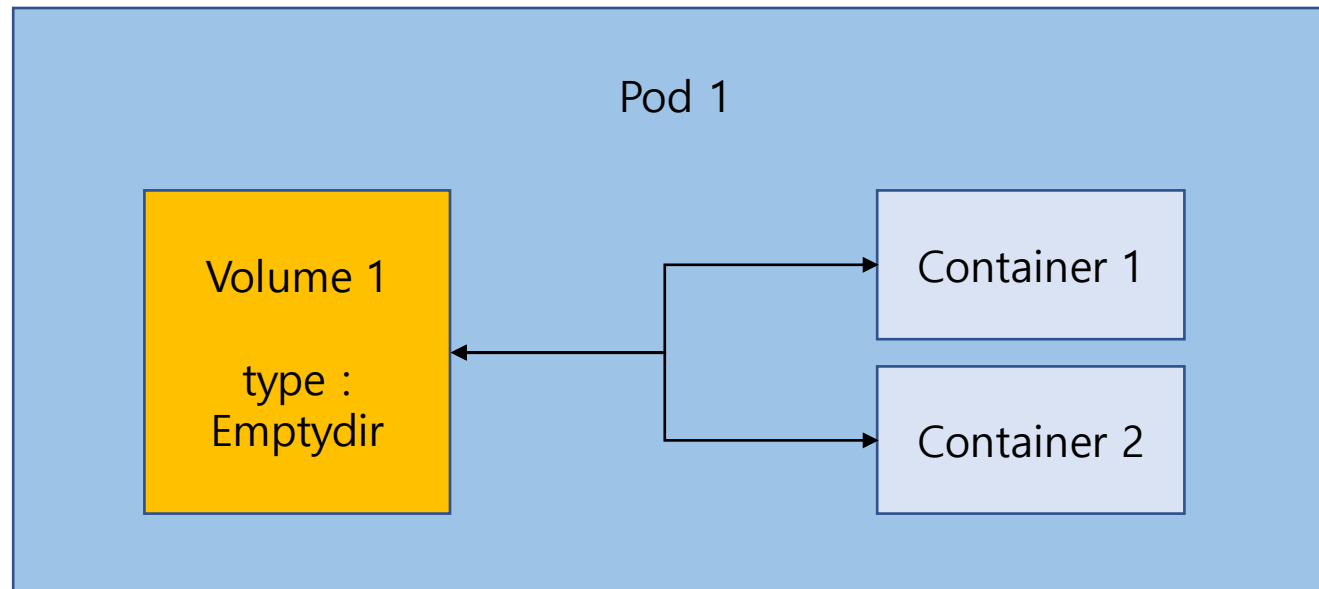
Volume 유형

- | emptyDir
- | hostPath
- | PV/PVC



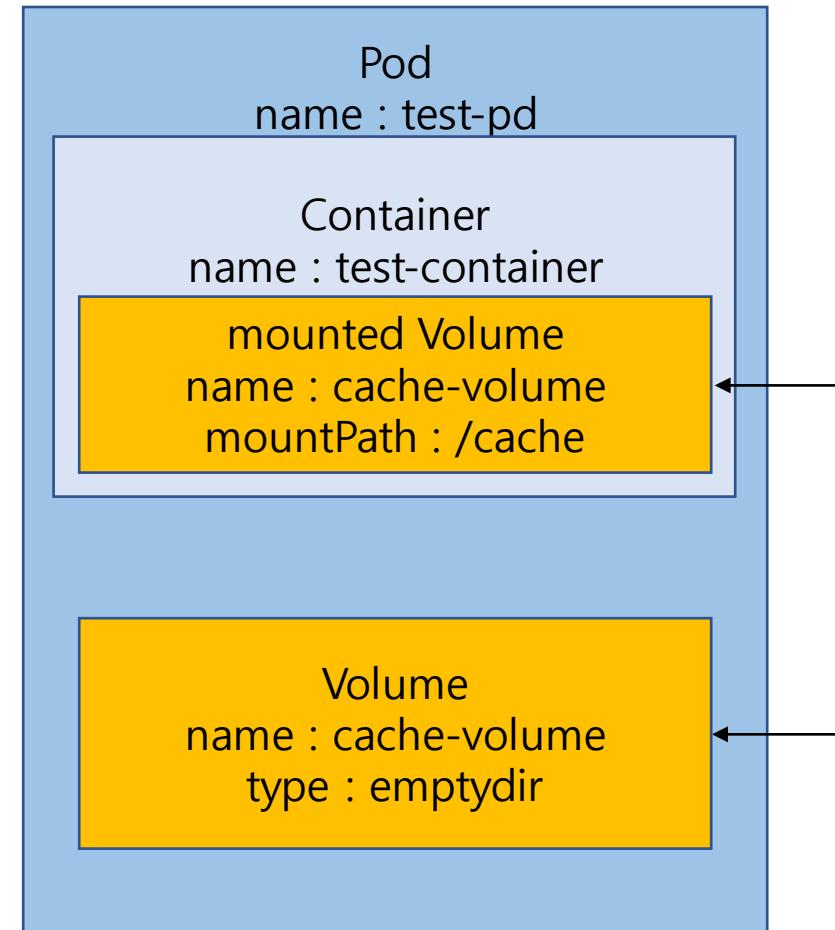
Emptydir

- Pod가 생성될 때 함께 생성되고,
Pod가 삭제될 때 함께 삭제되는 임시 Volume



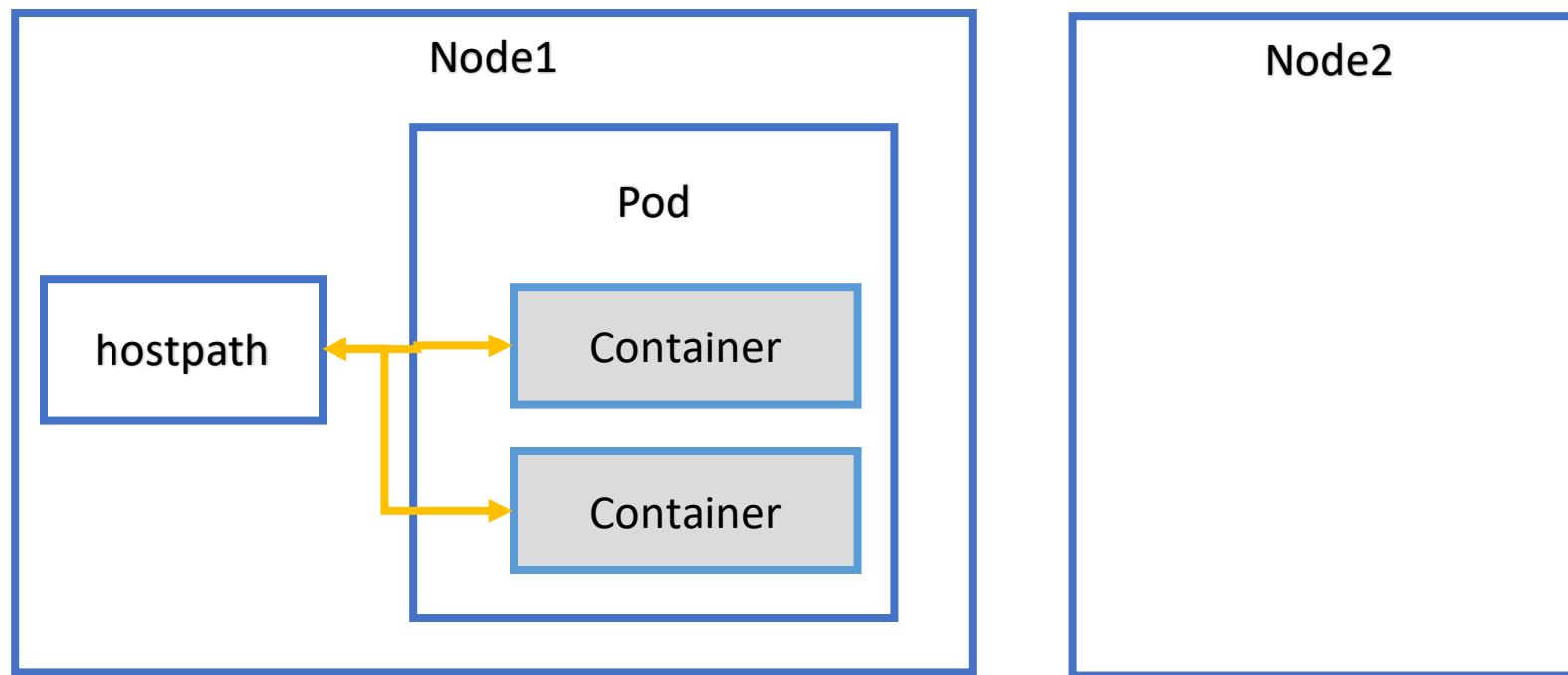
Emptydir YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```



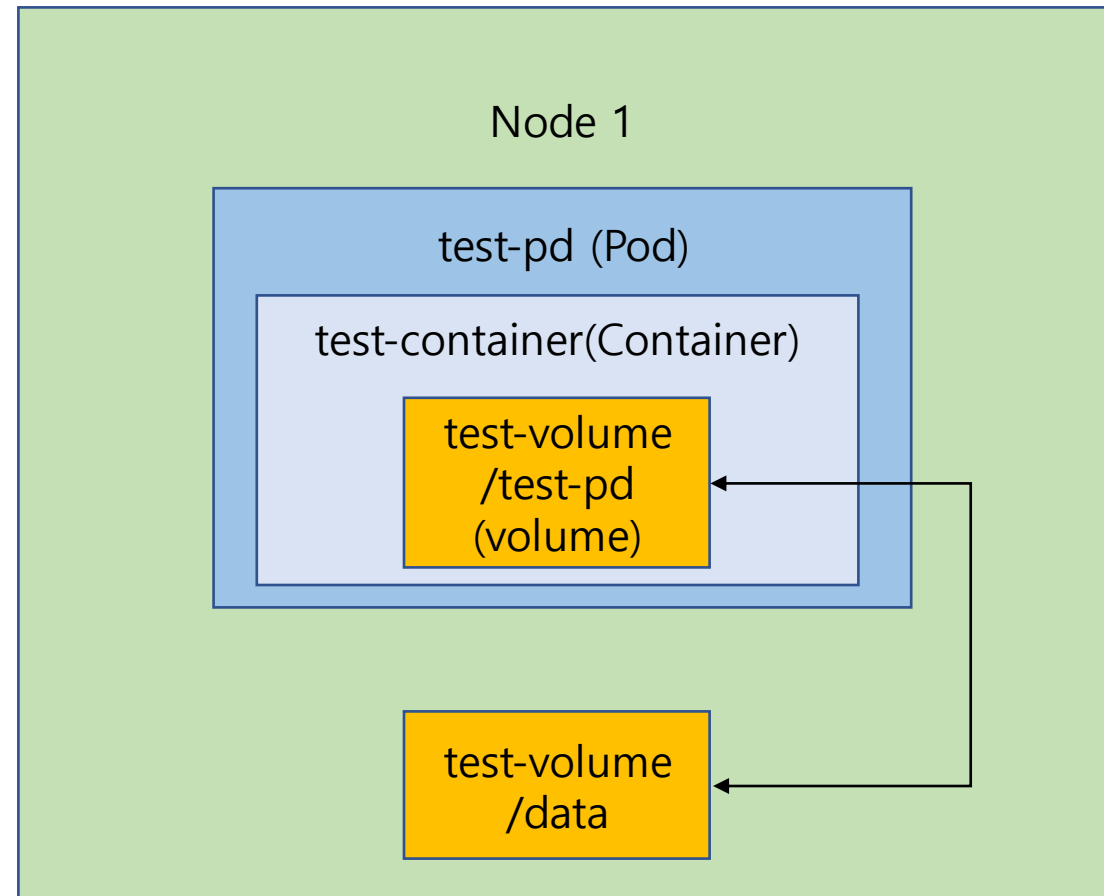
hostPath

- 호스트 노드의 경로를 Pod에 마운트하여 함께 사용하는 유형의 Volume

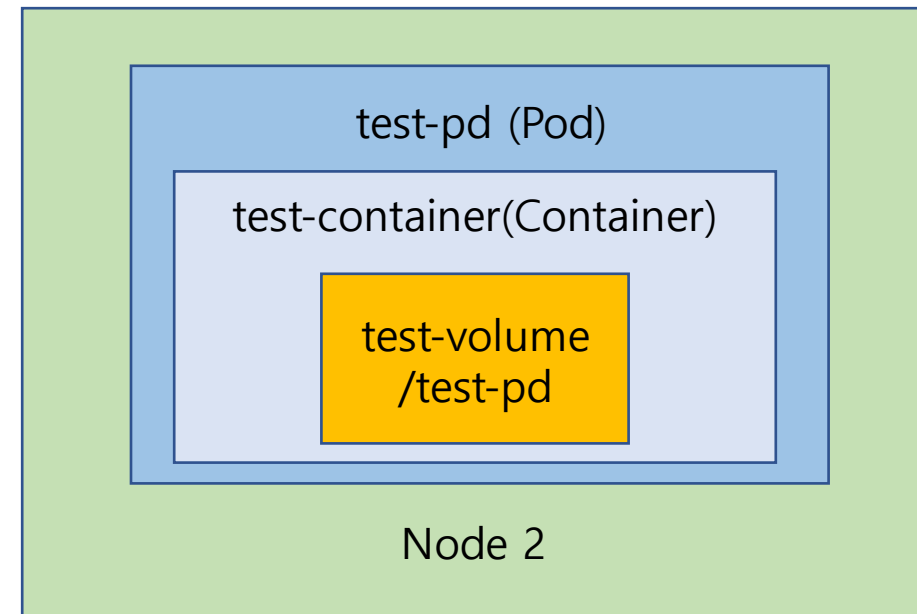
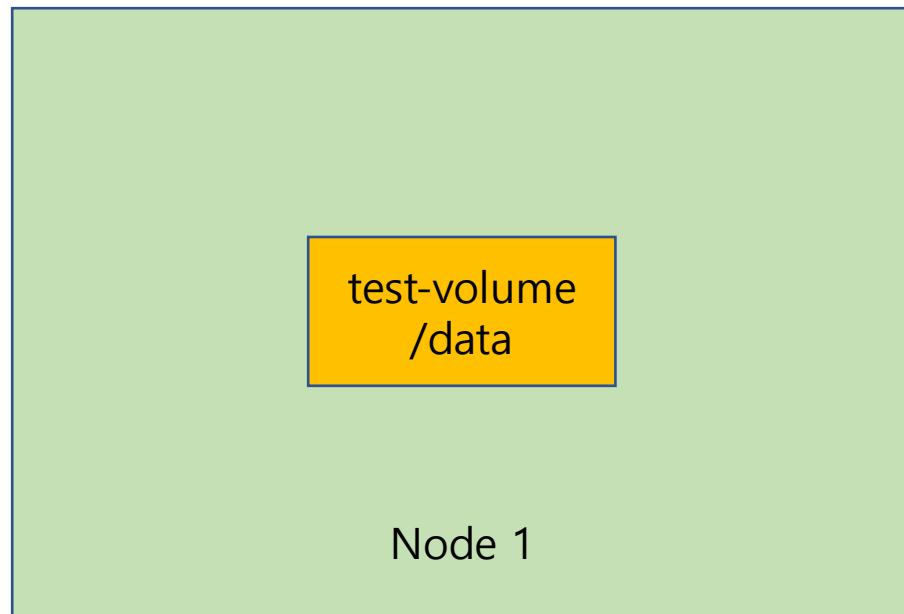


hostPath YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    hostPath:
      path: /data
```



hostPath - issue



PV

- Persistent Volume으로써 Volume 자체를 의미
- 클러스터 내부에서 Object처럼 관리 가능
- Pod와는 별도로 관리

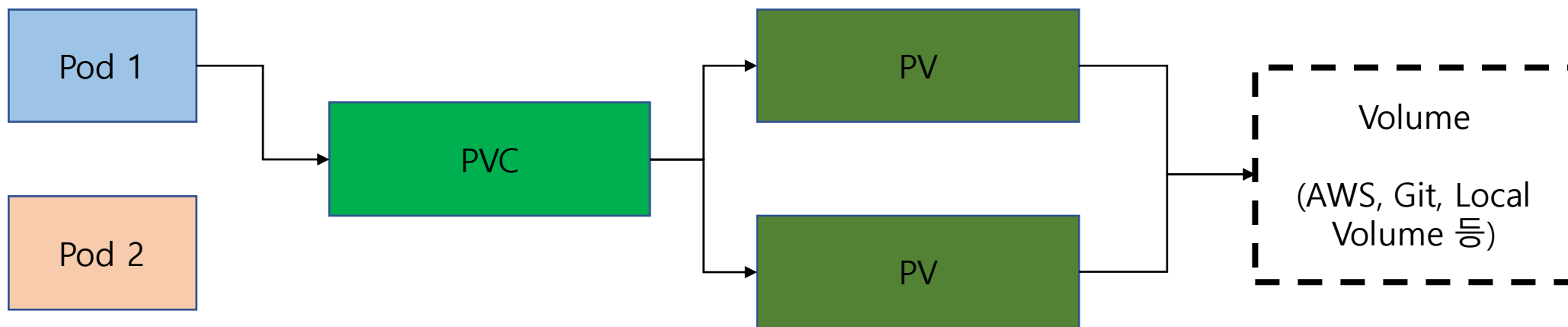
PVC

- PVC : PersistentVolumeClaim
- 사용자가 PV에 하는 요청
- Kubernetes는 Volume을 Pod에 직접 할당하는 방식이 아닌 중간에 PVC를 둬으로써 Pod와 스토리지를 분리

PV & PVC -1

- 클라우드 서비스에서 제공해주는 Volume 서비스를 이용할 수도 있고, 사설에 직접 구축 되어있는 스토리지를 사용 가능
- Pod에 직접 연결하지 않고 PVC를 통해서 사용

PV & PVC -2



PV YAML

- 로컬 디스크의 5G를 pv1이라는 이름으로 PV를 생성하는 설정의 yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  capacity:
    storage: 5G
  accessModes:
    - ReadWriteOnce
  local:
    path: /node1
```

PVC YAML

- 2G의 Volume 사용을 요청하는
pvc2라는 이름의 PVC를 생성
하는 설정의 yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc2
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2G
```

ConfigMap

- ㅣ 컨테이너에서 필요한 환경설정을
컨테이너와 분리하여 제공하는 Object
- ㅣ ConfigMap은 환경설정 정보들을 저장 해놓는 저장소 역할
- ㅣ 키/값 형태로 저장
- ㅣ ConfigMap의 생성 방식
 - ▶ Literal(문자) 방식
 - ▶ 파일 방식

Secret

- | 보안 관련 설정 값들을 저장하는 Object
- | 패스워드, API 키, 인증서 파일들을 저장
- | 사용방식은 ConfigMap과 동일

Literal - ConfigMap

- | Key와 Value로 구성
- | 필요한 상수를 정의

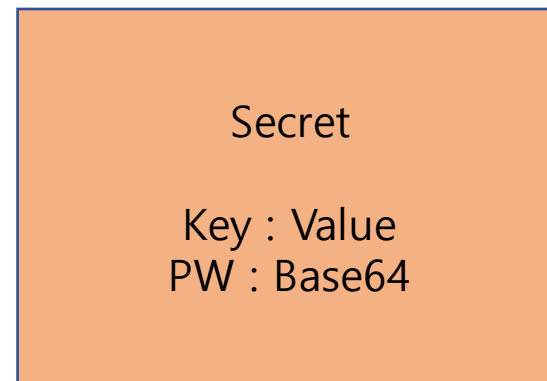
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cm-dev
data:
  SSH: 'false'
  User: dev
```

ConfigMap

Key : Value
SSH : False
user : dev

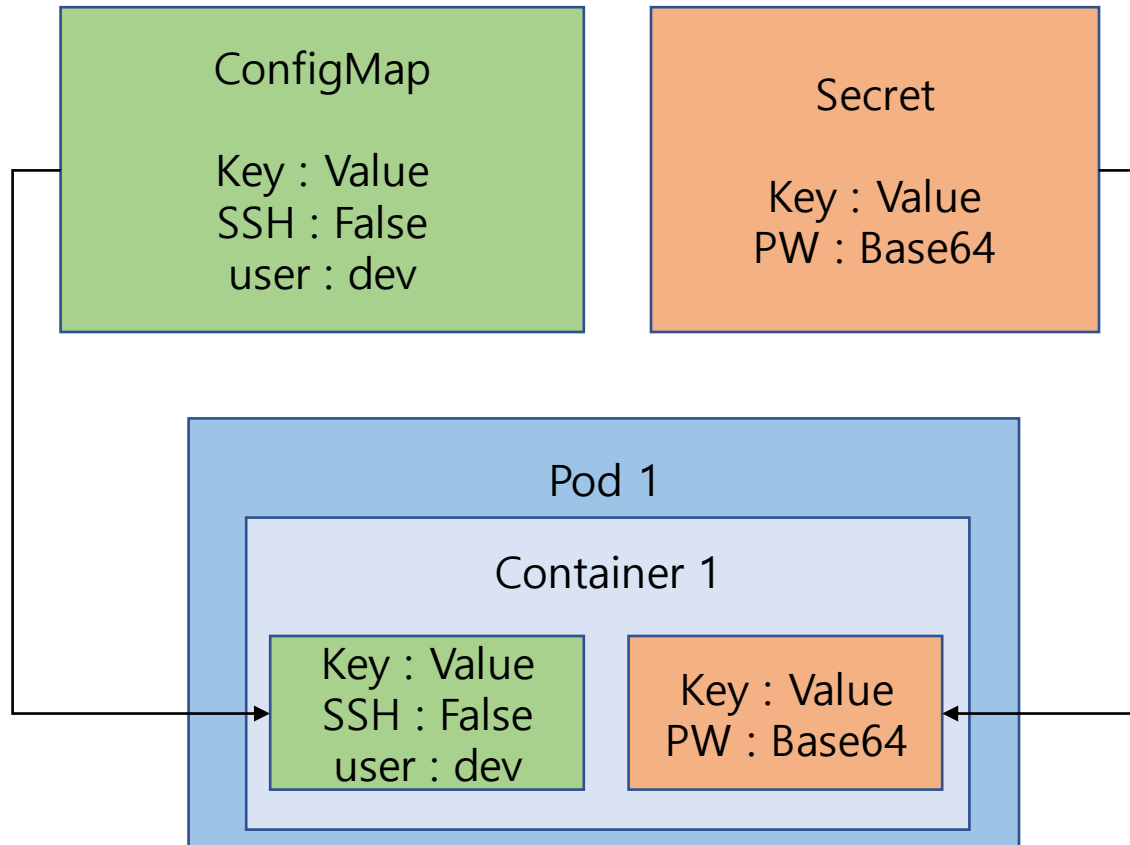
Literal - Secret

- | Secret의 Value에는 Base64 인코딩을 해서 저장 필요
- | 보안적인 요소는 아니며 Secret의 룰
- | Pod에 입력될 때는 자동으로 디코딩되어 원래 값이 출력



```
apiVersion: v1
kind: Secret
metadata:
  name: sc-dev
data:
  Key: MTIzNDU=
```


Literal – ConfigMap & Secret



```
apiVersion: v1
kind: Pod
metadata:
  name: cs-pod
spec:
  containers:
    - name: container
      image: nginx
      envFrom:
        - configMapRef:
            name: cm-dev
        - secretRef:
            name: sc-dev
```

File - ConfigMap

I cmfile.txt 를 이용하여 cm-file이라는 ConfigMap을 만드는 명령

▶ `kubectl create configmap cm-file --from-file=./cmfile.txt`

cmfile.txt

SSH : False

user : dev

File - Secret

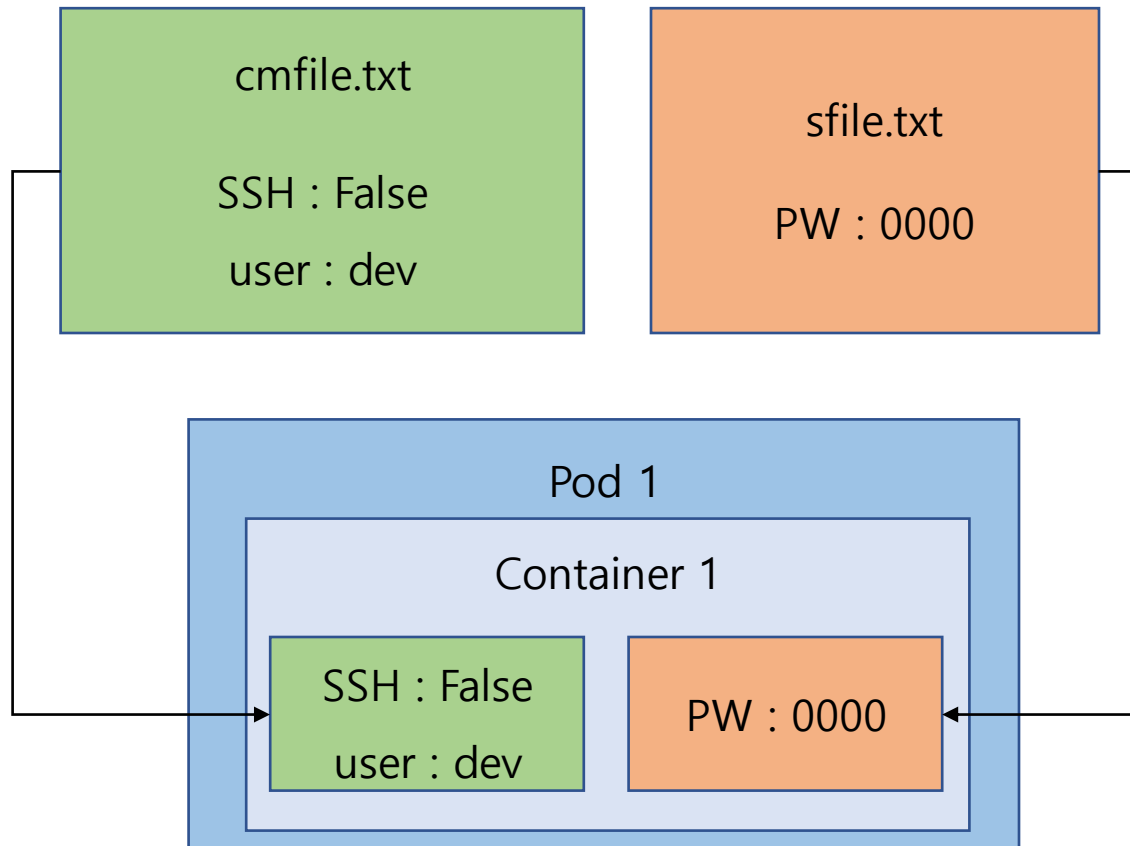
l sfile.txt 를 이용하여 sec-file이라는 Secret 을 만드는 명령

▶ `kubectl create secret generic sec-file --from-file=./sfile.txt`

sfile.txt

PW : 0000

Literal – ConfigMap & Secret



```
apiVersion: v1
kind: Pod
metadata:
  name: filecs-pod
spec:
  containers:
    - name: container
      image: nginx
      env:
        - name: cfile
          valueFrom:
            configMapKeyRef:
              name: fcm
              key: cfile.txt
        - name: sfile
          valueFrom:
            secretKeyRef:
              name: fsc
              key: sfile.txt
```

Chapter 6

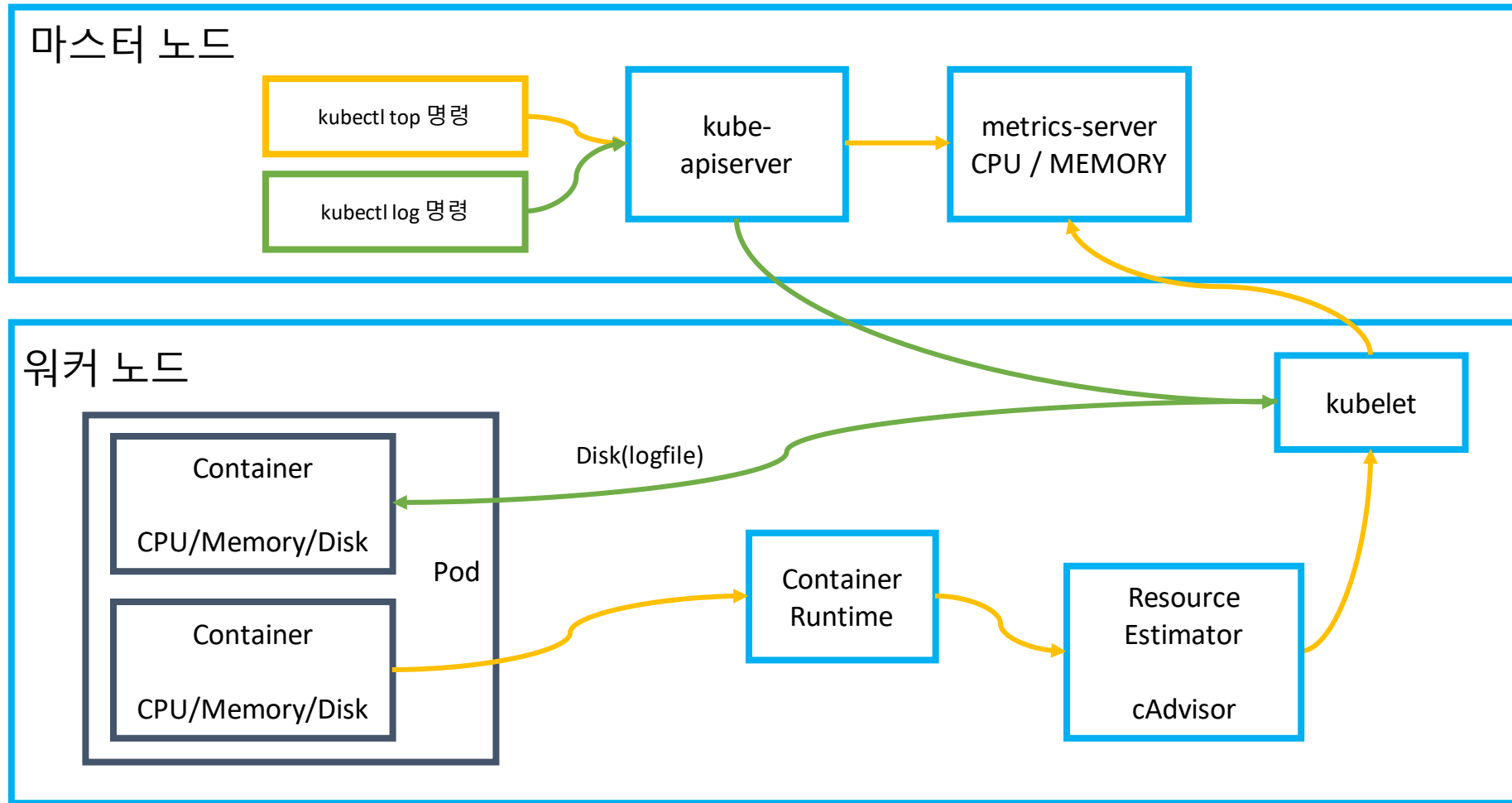
I 컨테이너 관리 - Dashboard, Autoscaling

로깅 및 모니터링

- 로깅과 모니터링은 시스템이 안정적으로 구동이 되는지를 확인하기 위한 감시체계입니다.



로깅 및 모니터링



Kubectl top

kubectl top 명령

kubectl top pod

: Default Namespace내 pod의 CPU, MEMORY 정보를 확인합니다.

kubectl top pod --all-namespaces

: 모든 Namespace내 Pod의 CPU, MEMORY 정보를 확인합니다.

Kubectl logs

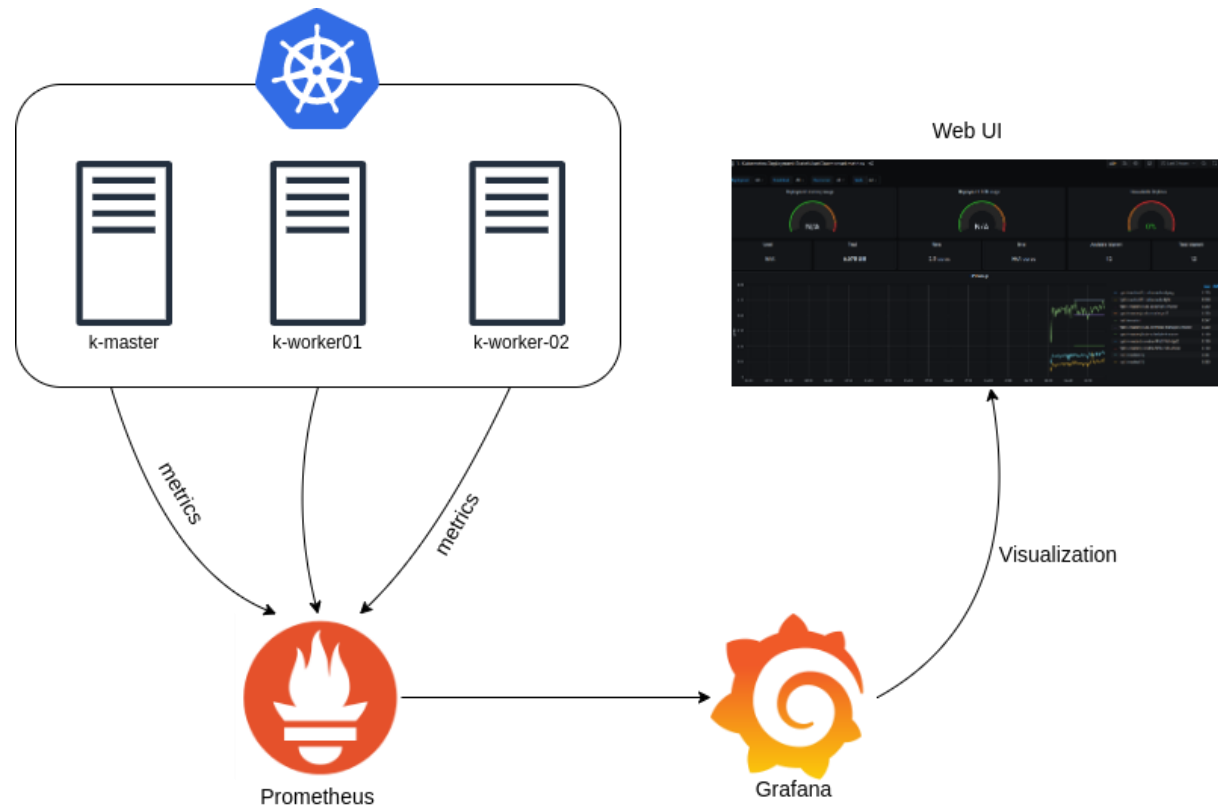
kubectl logs 명령

kubectl logs <Pod명>
: <Pod명> 이름의 Pod의 로그를 조회합니다.

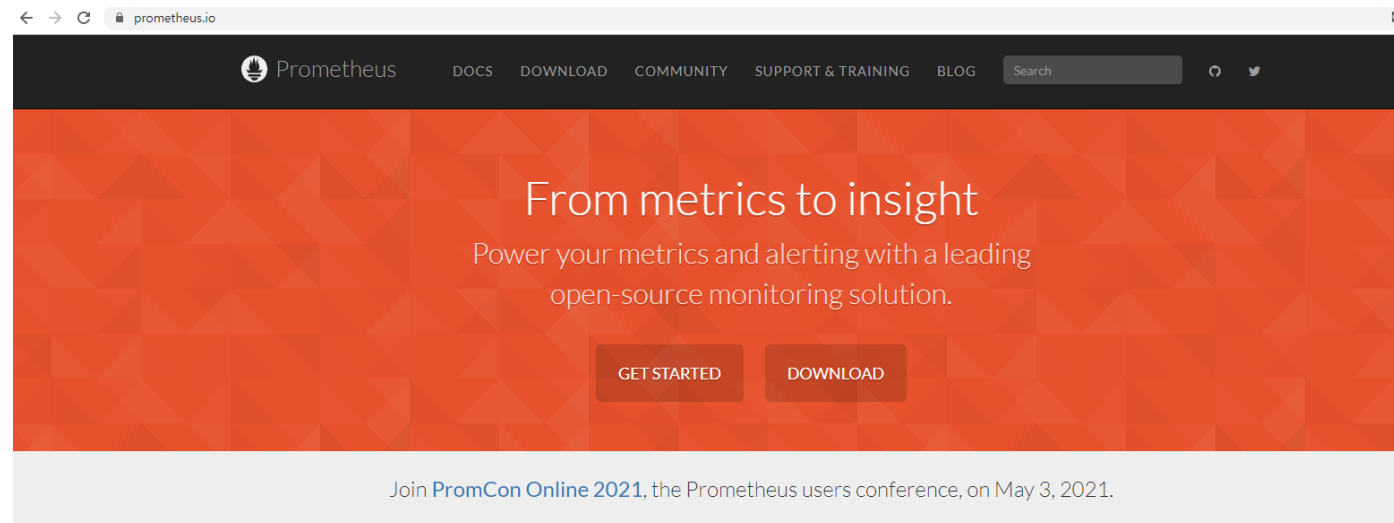
kubectl logs -f <Pod명>
: <Pod명> 이름의 Pod의 로그를 실시간으로 조회합니다.

로깅, 모니터링 추가 툴

I Kubernetes 모니터링 툴들



Prometheus



Dimensional data

Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.



Powerful queries

PromQL allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.



Great visualization

Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.



Efficient storage

Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.



Simple operation

Each server is independent for



Precise alerting

Alerts are defined based on



Many client libraries

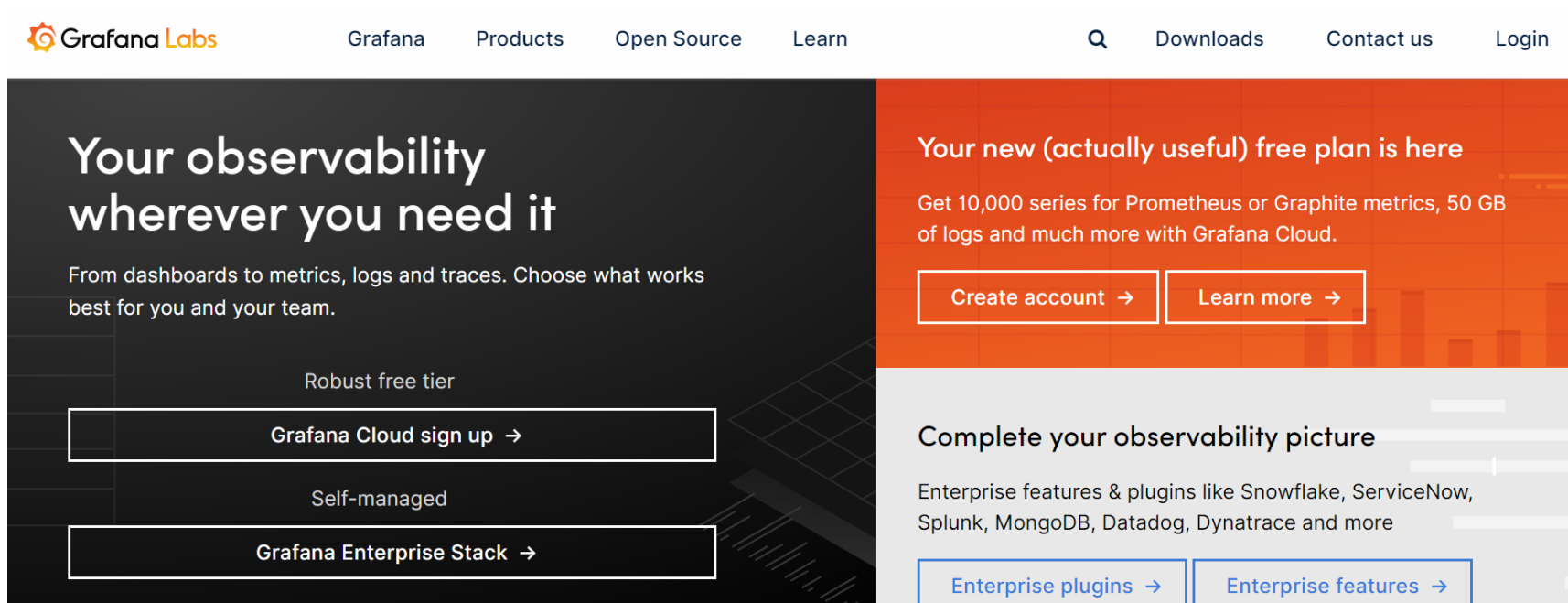
Client libraries allow easy




Many integrations

Existing exporters allow bridging of

Grafana



The screenshot shows the Grafana Labs website homepage. The header features the Grafana Labs logo on the left and navigation links for Grafana, Products, Open Source, Learn, a search icon, Downloads, Contact us, and Login on the right. The main content is divided into two columns. The left column has a dark background with the headline 'Your observability wherever you need it' and a sub-headline 'From dashboards to metrics, logs and traces. Choose what works best for you and your team.' Below this are two options: 'Robust free tier' with a 'Grafana Cloud sign up →' button, and 'Self-managed' with a 'Grafana Enterprise Stack →' button. The right column has an orange background with the headline 'Your new (actually useful) free plan is here' and a sub-headline 'Get 10,000 series for Prometheus or Graphite metrics, 50 GB of logs and much more with Grafana Cloud.' Below this are two buttons: 'Create account →' and 'Learn more →'. At the bottom of the right column, on a light gray background, is the section 'Complete your observability picture' with a sub-headline 'Enterprise features & plugins like Snowflake, ServiceNow, Splunk, MongoDB, Datadog, Dynatrace and more' and two buttons: 'Enterprise plugins →' and 'Enterprise features →'.

Grafana Labs Grafana Products Open Source Learn  Downloads Contact us Login

Your observability wherever you need it

From dashboards to metrics, logs and traces. Choose what works best for you and your team.

Robust free tier

[Grafana Cloud sign up →](#)

Self-managed

[Grafana Enterprise Stack →](#)

Your new (actually useful) free plan is here

Get 10,000 series for Prometheus or Graphite metrics, 50 GB of logs and much more with Grafana Cloud.

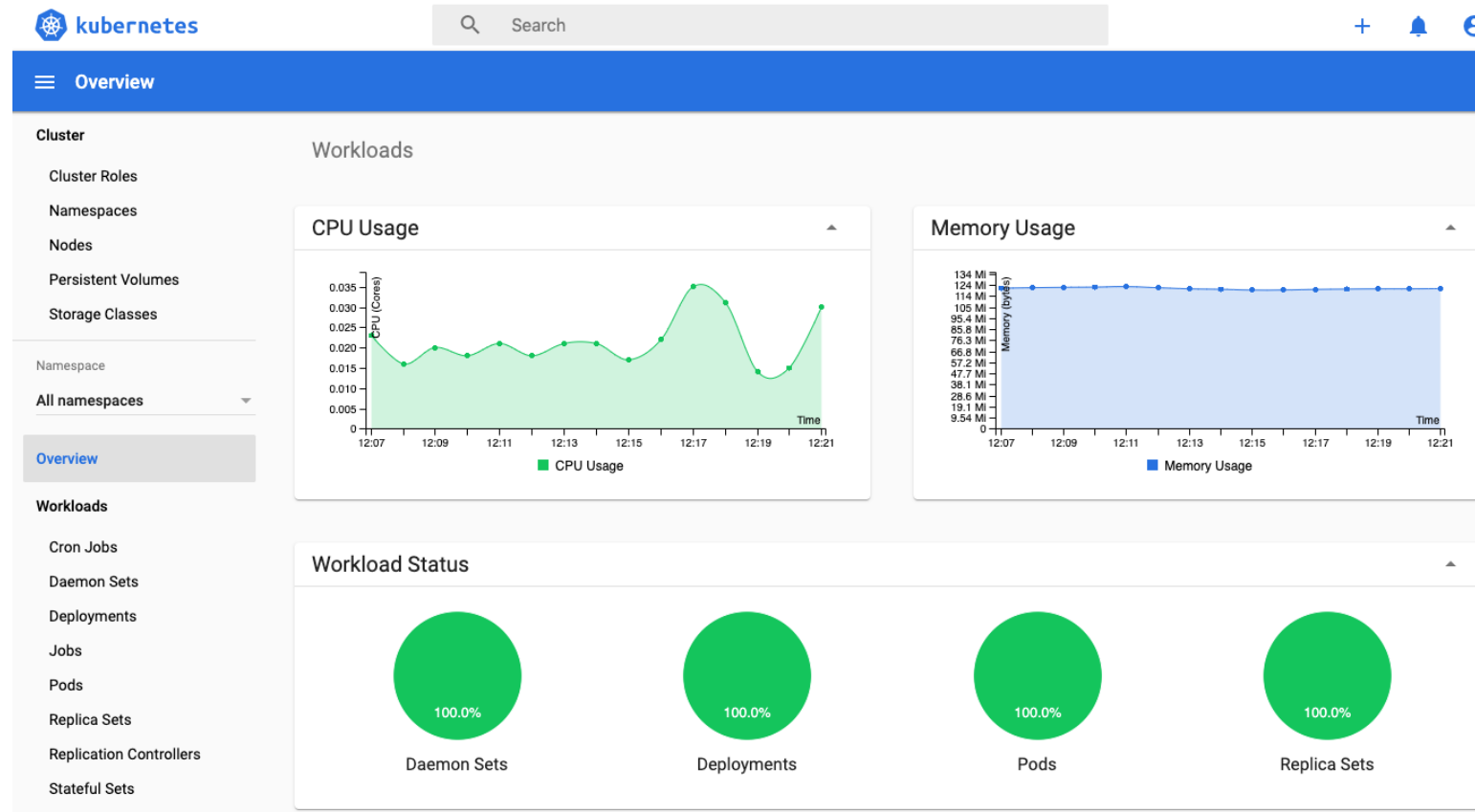
[Create account →](#) [Learn more →](#)

Complete your observability picture

Enterprise features & plugins like Snowflake, ServiceNow, Splunk, MongoDB, Datadog, Dynatrace and more

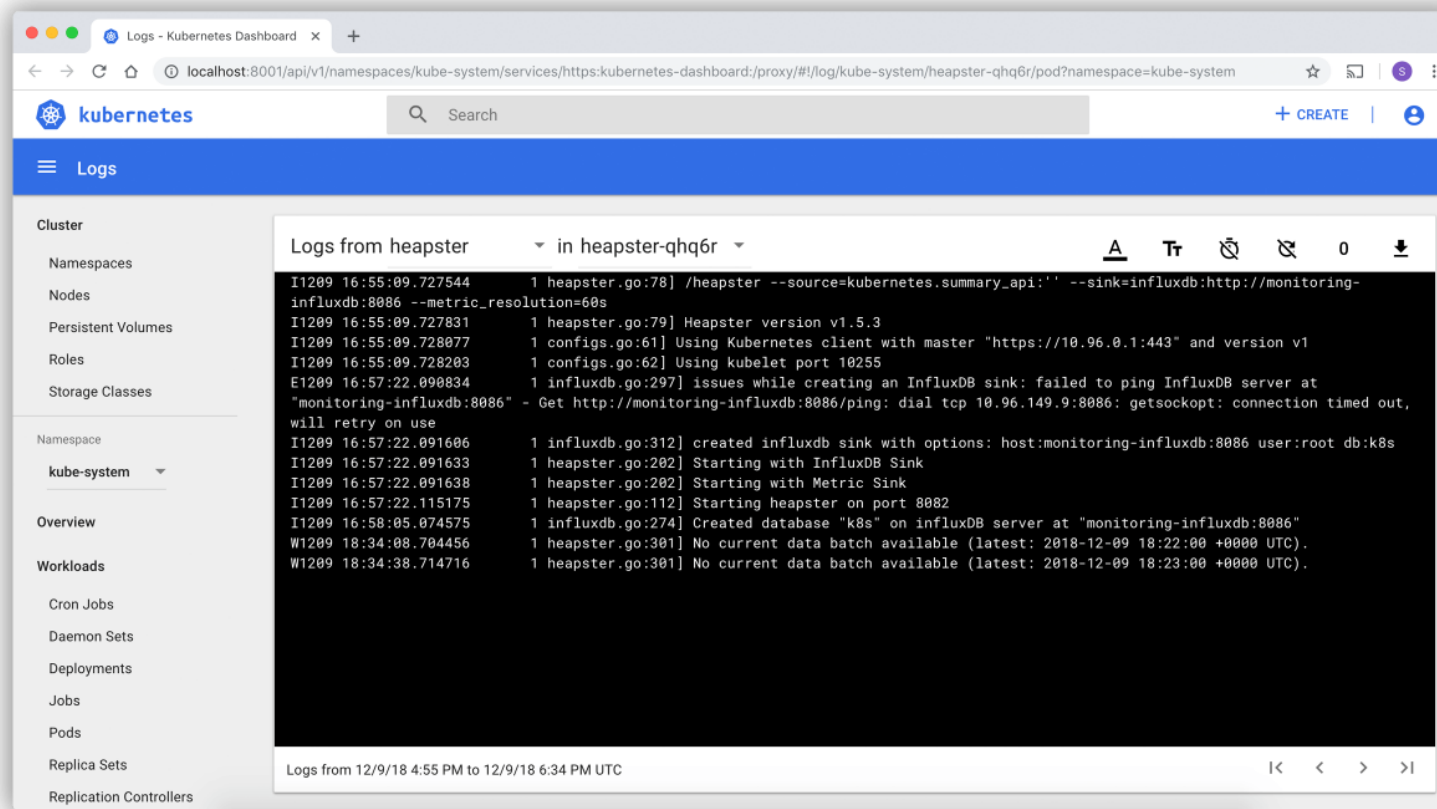
[Enterprise plugins →](#) [Enterprise features →](#)

Kubernetes 대쉬보드



Kubernetes 대쉬보드

I 로그뷰어



트러블슈팅

- 서비스되고 있거나 테스트중인 애플리케이션 또는 노드에 정상적인 작동이 되지않을 때, 문제 해결을 위한 방법들을 소개합니다.



트러블슈팅

I 오타 (Typo)

- ▶ 명령어에서의 오타
- ▶ 파일명, 오브젝트명, 컨트롤러명, 대쉬기호 등 수 많은 오타가 발생할 여지가 있습니다
- ▶ Yaml 파일에서의 오타
- ▶ 들어쓰기 실수, 파일명, 오브젝트명, 이미지명 등등 많은 오타가 있을 수 있습니다

트러블슈팅

I get / describe / logs / exec 명령

- ▶ get 명령으로 오브젝트 및 컨트롤러의 간단한 상태를 조회하고 이상 상태 발생시 해결합니다.
- ▶ describe 명령은 오브젝트 및 컨트롤러의 더 상세한 내용을 조회할 수 있습니다.
- ▶ logs 명령으로 Pod의 로그 정보를 조회하여 문제를 해결합니다.
- ▶ exec 명령으로 실행중인 Pod 에 접속하여 직접 로그 및 상태를 확인 할 수 있습니다.

Thank you