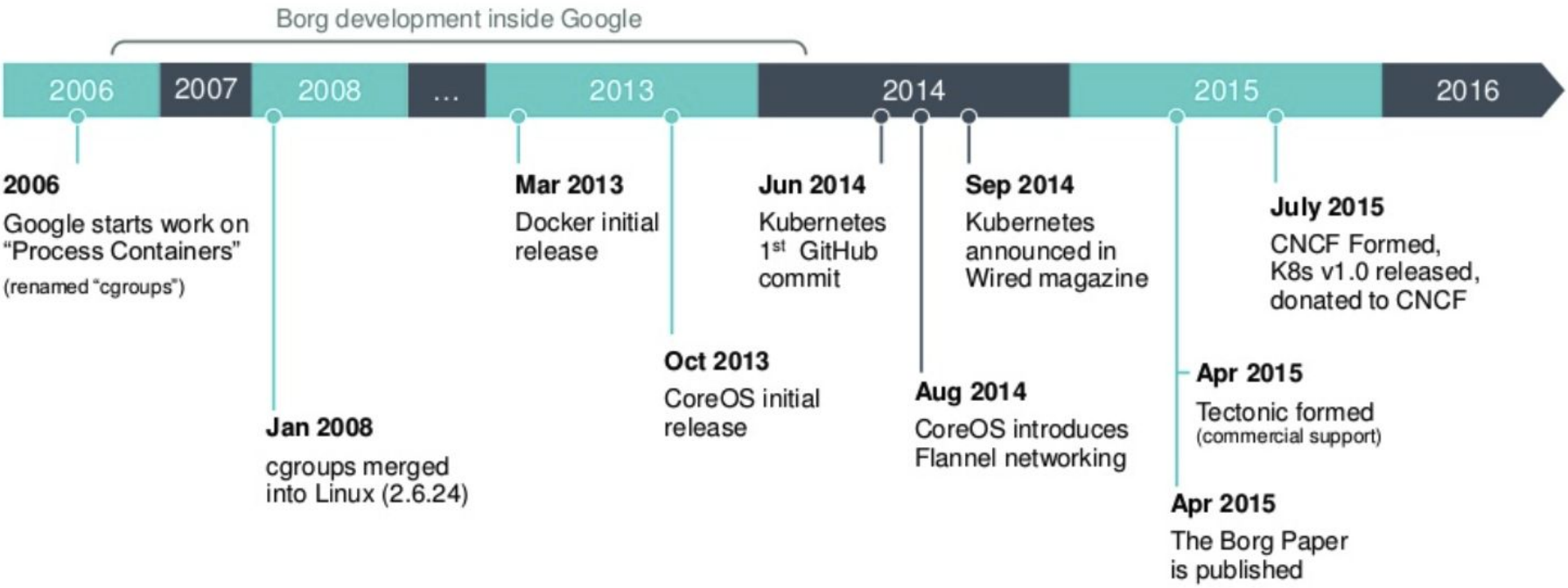
Kubernetes Walkthrough

2021. 4. 28 / Sangwon Lee

목차

1. 준비

Kubernetes 소개 (1/4) - History



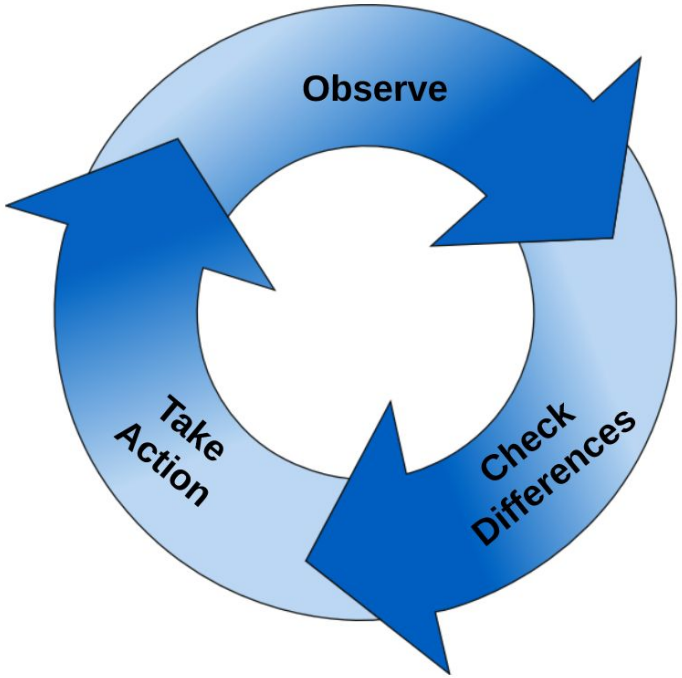
https://www.slideshare.net/egg9/kubernetes-introduction

Kubernetes 소개 (2/4) - Why?

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling or removing containers based on balancing workloads across your infrastructure
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

→ **Container Orchestration**

Kubernetes 소개 (3/4) - Core Mechanism



Desired-State and Control-loops

1. Observe
What is the desired state of our objects?

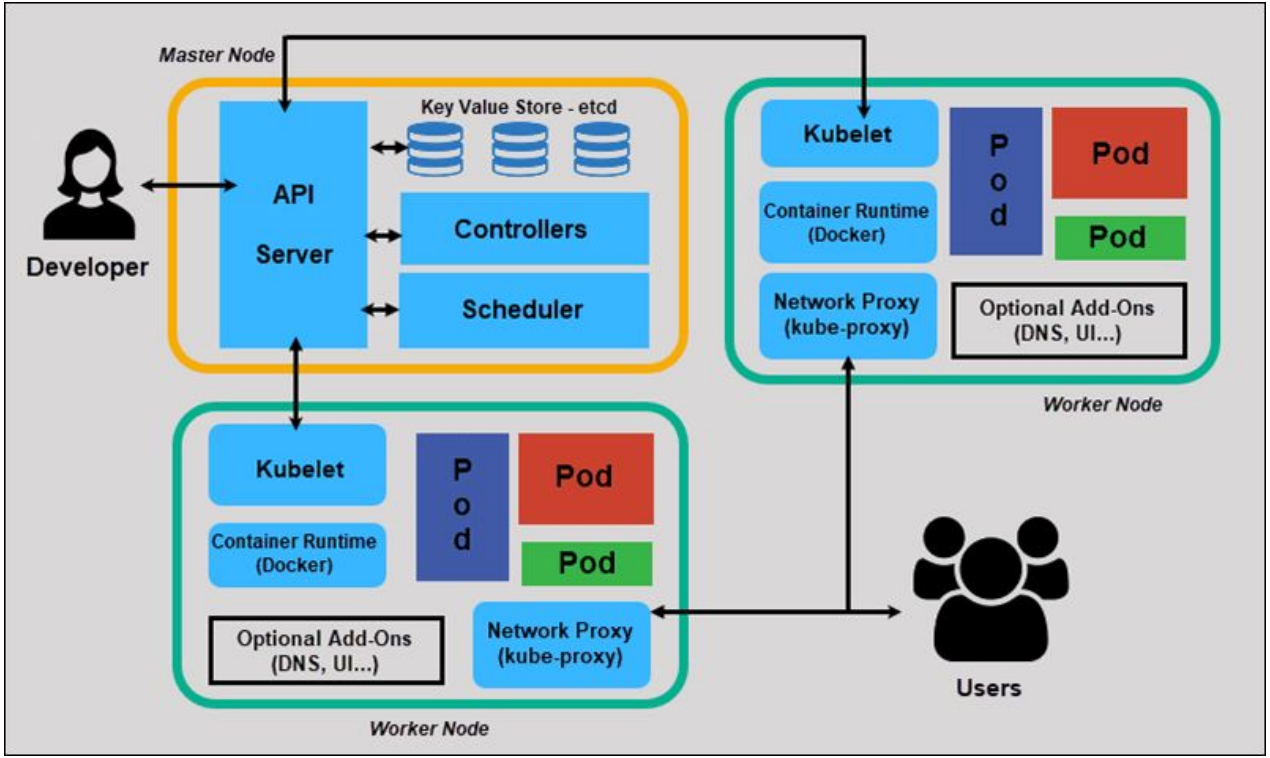
2. Check Differences
What is the current state of our objects and the differences between our desired state?

3. Take Action
Make the current state look like the desired state.

4. Repeat
Repeat this over and over again.

https://theithollow.com/2019/09/16/kubernetes-desired-state-and-control-loops/

Kubernetes 소개 (4/4) - Architecture



https://medium.com/@mohan08p/simplified-kubernetes-architecture-3febe12480eb

클러스터 할당 및 접근 설정 (1/3) - kube config

```
➜ cat ~/.kube/config
apiVersion: v1
kind: Config
clusters:
- name: my-dev-cluster
  cluster:
    insecure-skip-tls-verify: true
    server: https://my-dev-cluster-master-1.example.com:6443

users:
- name: my-dev-creds
  user:
    token: eyJhbGciOiJSUzI1NiIsImtpZCI6…

contexts:
- context:
    cluster: my-dev-cluster
    user: my-dev-creds
    namespace: ingress-nginx
  name: my-dev-ctx
current-context: my-dev-ctx
preferences: {}
...
```

클러스터 할당 및 접근 설정 (2/3) - kubectl 설치

https://kubernetes.io/docs/tasks/tools/

```
# MacOS
➜ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl"

➜ kubectl version
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.4",
GitCommit:"d360454c9bcd1634cf4cc52d1867af5491dc9c5f", GitTreeState:"clean", BuildDate:"2020-11-14T14:49:35Z",
GoVersion:"go1.15.5", Compiler:"gc", Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.11",
GitCommit:"d94a81c724ea8e1ccc9002d89b7fe81d58f89ede", GitTreeState:"clean", BuildDate:"2020-03-12T21:00:06Z",
GoVersion:"go1.12.17", Compiler:"gc", Platform:"linux/amd64"}
```

클러스터 할당 및 접근 설정 (3/3) - kubectx 설치 (추천)

https://github.com/ahmetb/kubectx

`kubectx` + `kubens` : Power tools for kubectl

release v0.9.3 | github stars 10k | Go implementation (CI) passing | written in bash

This repository provides both `kubectx` and `kubens` tools. Install →

> 🥳🆕 **NEWS:** With v0.9.0 `kubectx` and `kubens` **are now rewritten in Go.** (Don't worry, our lovely **bash** versions are still available!) Please test this new Go binaries by downloading them from **Releases →**

`kubectx` helps you switch between clusters back and forth:

```
➜  ~ kubectx
coffee
minikube
test
➜  ~ kubectx coffee
Switched to context "coffee".
➜  ~ kube
```

클러스터 할당 및 접근 설정 (3/3) - kubectx 설치 (추천)

```
➜  kubectx
Switched to context "my-dev-ctx".

➜  kubens
  hello-app-beta
> hello-app-production
  2/16
Context "my-dev-ctx" modified.
Active namespace is "hello-app-production".

➜  kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
hello-app-6dd9cd8bcc-gk9m6   3/3     Running   0          20h
hello-app-6dd9cd8bcc-nkdbb   3/3     Running   0          20h
```

2. 설정파일 작성

직접 작성 (1/3) - 오브젝트(Object)의 구성

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

이 오브젝트를 생성하기 위한 Kubernetes API 버전

오브젝트 종류 - Pod, Service, Ingress 등

오브젝트를 식별할 수 있는 정보 제공
- name 필수
- namespace 옵션 (기본 default)

오브젝트의 desired state를 기술
오브젝트 kind 마다 스펙이 다름

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

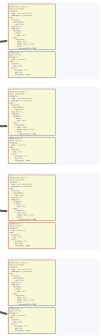
직접 작성 (2/3) - 한 파일에 여러 오브젝트 기술

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 8000
---
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
```

직접 작성 (3/3) - Phase 별로 구성

my-app1

- app-dev.yaml
- app-sandbox.yaml
- app-cbt.yaml
- app-production.yaml



my-app2

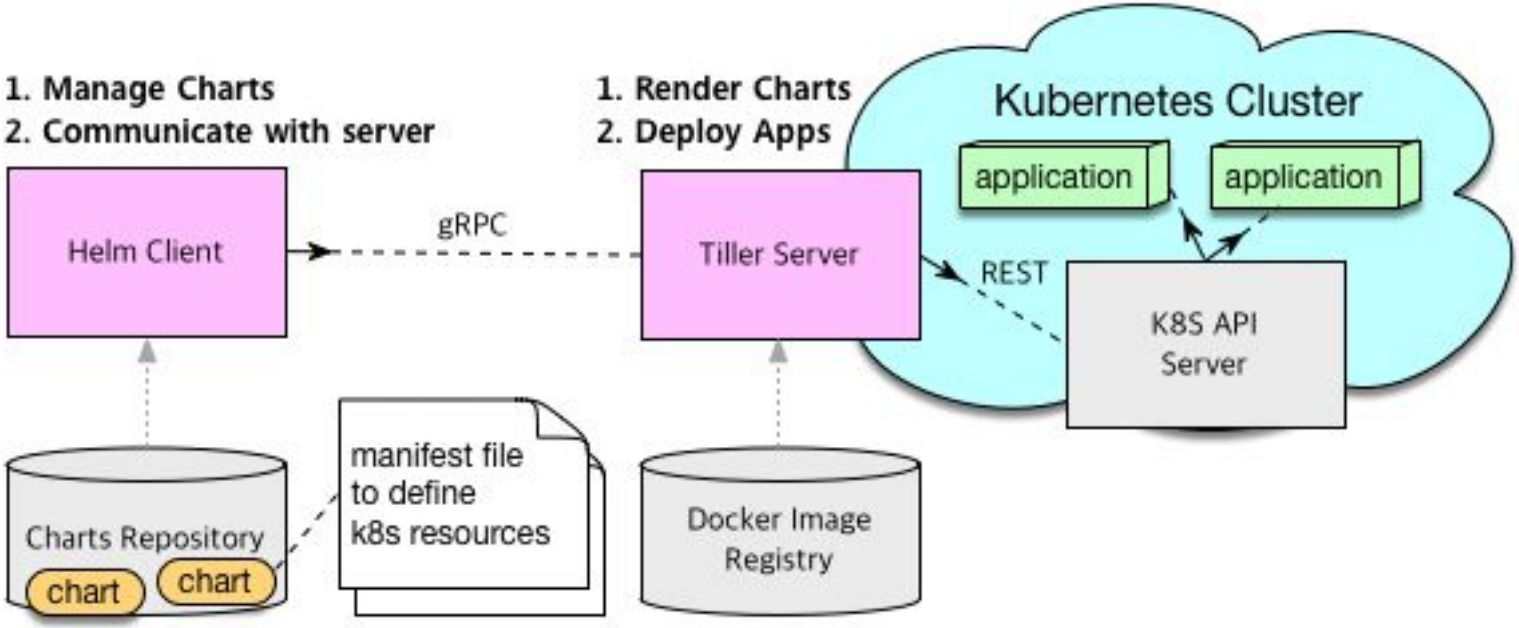
...

- sandbox
  - deployment.yaml
  - service.yaml
  - secret.yaml
- production
  - deployment.yaml
  - service.yaml
  - secret.yaml

```
➜ $ cd my-app1
➜ my-app1$ kubectl apply -f ./app-sandbox.yaml
```

```
➜ $ cd my-app2/sandbox
➜ my-app2/sandbox$ kubectl apply -f .

or

➜ my-app2$ kubectl apply -f ./sandbox
```

Helm (1/3) - How it works



Helm (1/3) - 그러나 우리는 그냥 템플릿 도구로써

```
data := TodoPageData{
    PageTitle: "My TODO list",
    Todos: []Todo{
        {Title: "Task 1", Done: false},
        {Title: "Task 2", Done: true},
        {Title: "Task 3", Done: true},
    },
}
```

```
<h1>{{.PageTitle}}</h1>
<ul>
    {{range .Todos}}
        {{if .Done}}
            <li class="done">{{.Title}}</li>
        {{else}}
            <li>{{.Title}}</li>
        {{end}}
    {{end}}
</ul>
```

https://gowebexamples.com/templates/

Helm (2/4) - Template 생성

```
➜ k8s-test$ helm create my-app
Creating my-app

➜ k8s-test$ ls -al
total 0
drwxr-xr-x    3 sangwonl  staff    96 Apr  1 17:35 .
drwxr-xr-x  103 sangwonl  staff  3296 Apr  1 17:35 ..
drwxr-xr-x    7 sangwonl  staff   224 Apr  1 17:35 my-app

➜ k8s-test$ tree my-app
my-app
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 9 files
```

Helm (2/4) - Template 생성

```
➜ k8s-test$ helm create my-app
Creating my-app

➜ k8s-test$ ls -al
total 0
drwxr-xr-x    3 sangwonl  staff
drwxr-xr-x  103 sangwonl  staff
drwxr-xr-x    7 sangwonl  staff

➜ k8s-test$ tree my-app
my-app
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 9 files
```

```
➜ k8s-test$ cat my-app/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "my-app.fullname" . }}
  labels:
{{ include "my-app.labels" . | indent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app.kubernetes.io/name: {{ include "my-app.name" . }}
      app.kubernetes.io/instance: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app.kubernetes.io/name: {{ include "my-app.name" . }}
        app.kubernetes.io/instance: {{ .Release.Name }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          securityContext:
            {{- toYaml .Values.securityContext | nindent 12 }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
    ...
    ..
```

Helm (2/4) - Template 생성

```
➜ k8s-test$ helm create my-app
Creating my-app

➜ k8s-test$ ls -al
total 0
drwxr-xr-x    3 sangwonl  staff    96 Apr  1 17:35 .
drwxr-xr-x  103 sangwonl  staff  3296 Apr  1 17:35 .
drwxr-xr-x    7 sangwonl  staff   224 Apr  1 17:35 my

➜ k8s-test$ tree my-app
my-app
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 9 files
```

```yaml
replicaCount: 1

image:
  repository: nginx
  tag: stable
  pullPolicy: IfNotPresent

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

service:
  type: ClusterIP
  port: 80

ingress:
  enabled: false
  annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
  hosts:
    - host: chart-example.local

...
..
```

Helm (3/4) - Template 렌더

```
➜ my-app$ helm template . -f values.yaml
---
# Source: my-app/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: release-name-my-app
  labels:
    app.kubernetes.io/name: my-app
    helm.sh/chart: my-app-0.1.0
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.0"
    app.kubernetes.io/managed-by: Tiller
---
# Source: my-app/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: release-name-my-app
  labels:
    app.kubernetes.io/name: my-app
    helm.sh/chart: my-app-0.1.0
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.0"
...
..
```

Helm (3/4) - Template 렌더 (검증)

```
➜ my-app$ helm template . -f values.yaml
---
# Source: my-app/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: release-name-my-app
  labels:
    app.kubernetes.io/name: my-app
    helm.sh/chart: my-app-0.1.0
    app.kubernetes.io/instance: rele
    app.kubernetes.io/version: "1.0"
    app.kubernetes.io/managed-by: Ti
---
# Source: my-app/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: release-name-my-app
  labels:
    app.kubernetes.io/name: my-app
    helm.sh/chart: my-app-0.1.0
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.0"
...
..
```

```
➜ my-app$ helm template . -f values.yaml | kubectl apply -f - \
    --dry-run=client \
    --validate=true

serviceaccount/release-name-my-app created (dry run)
service/release-name-my-app created (dry run)
pod/release-name-my-app-test-connection created (dry run)
deployment.apps/release-name-my-app created (dry run)
```

Helm (3/4) - Template 렌더 (적용)

```
➜ my-app$ helm template . -f values.yaml
---
# Source: my-app/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: release-name-my-app
  labels:
    app.kubernetes.io/name: my-app
    helm.sh/chart: my-app-0.1.0
    app.kubernetes.io/instance: rele
    app.kubernetes.io/version: "1.0"
    app.kubernetes.io/managed-by: Ti
---
# Source: my-app/templates/service.y
apiVersion: v1
kind: Service
metadata:
  name: release-name-my-app
  labels:
    app.kubernetes.io/name: my-app
    helm.sh/chart: my-app-0.1.0
    app.kubernetes.io/instance: release-name
    app.kubernetes.io/version: "1.0"
...
..
```

```
➜ my-app$ helm template . -f values.yaml | kubectl apply -f -
serviceaccount/release-name-my-app created
service/release-name-my-app created
pod/release-name-my-app-test-connection created
deployment.apps/release-name-my-app created

➜ my-app$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
my-app-5dfb54954b-b7rvt     1/1     Running   0          10s
my-app-5dfb54954b-n6752     1/1     Running   0          13s
```

Helm (4/4) - Phase 별로 구성

```
➜ k8s-test$ tree my-app
my-app
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── service.yaml
│   └── secrets.yaml
├── overrides
│   ├── dev.yaml
│   ├── sandbox.yaml
│   ├── cbt.yaml
│   └── production.yaml
└── values.yaml

3 directories, 12 files
```

```
➜ my-app$ helm template . -f values.yaml -f overrides/sandbox.yaml
---
# Source: my-app/templates/deployment.yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: my-app
  namespace: my-app-sandbox
spec:
  selector:
    matchLabels:
      app: my-app
  replicas: 2
  template:
...
..
```

Kustomize (1/4) - 생성

```
➜ my-app$ ls
deployment.yaml ingress.yaml    secret.yaml     service.yaml

➜ my-app$ kustomize create --autodetect

➜ my-app$ ls
deployment.yaml    ingress.yaml     kustomization.yaml     secret.yaml       service.yaml

➜ my-app$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- ingress.yaml
- secret.yaml
- service.yaml
```

Kustomize (2/4) - 패치

```
➜ my-app$ ls
deployment.yaml       ingress.yaml    kustomizat
patch-resources.yaml  secret.yaml     service.ya

➜ my-app$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- ingress.yaml
- secret.yaml
- service.yaml

images:
- name: sangwonl/my-app
  newTag: 1.58.1

patchesStrategicMerge:
- patch-resources.yaml
```

```
➜ my-app$ cat patch-resources.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 2
  template:
    spec:
      containers:
      - name: my-app
        resources:
          requests:
            cpu: 2000m
            memory: 2.5Gi
          limits:
            cpu: 2000m
            memory: 2.5Gi
...
..
```

Kustomize (2/4) - 패치 (JSON6902)

```
➜ my-app$ cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- ingress.yaml
- secret.yaml
- service.yaml

images:
- name: sangwonl/my-app
  newTag: 1.58.1

patchesStrategicMerge:
- patch-resources.yaml

patchesJson6902:
- path: patch-ingress.yaml
  target:
    group: extensions
    kind: Ingress
    name: my-app-ingress
    version: v1beta1
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
  name: my-app-ingress
spec:
  rules:
  - host: plz-patch-me.example.com
    http:
      paths:
      - backend:
          serviceName: my-app-service
          servicePort: http-port
        path: /
  tls:
  - hosts:
    - plz-patch-me.example.com
    secretName: tls-secret-example-com
```

```
➜ my-app$ cat patch-ingress.yaml
- op: replace
  path: /spec/rules/0/host
  value: my-app-sandbox.example.com
- op: replace
  path: /spec/tls/0/hosts/0
  value: my-app-sandbox.example.com
```

Kustomize (3/4) - Template 렌더

```
➜ my-app$ kustomize build
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-app
  name: my-app-service
spec:
  ports:
  - name: http-port
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: my-app
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-app
spec:
  replicas: 2
...
..
```

Kustomize (3/4) - Template 렌더 (검증)

```
➜ my-app$ kustomize build
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-app
  name: my-app-service
spec:
  ports:
  - name: http-port
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: my-app
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-app
spec:
  replicas: 2
...
..
```

```
➜ my-app$ kustomize build | kubectl apply -f - \
    --dry-run=client \
    --validate=true

serviceaccount/release-name-my-app created (dry run)
service/release-name-my-app created (dry run)
pod/release-name-my-app-test-connection created (dry run)
deployment.apps/release-name-my-app created (dry run)
```

Kustomize (3/4) - Template 렌더 (적용)

```
➜ my-app$ kustomize build
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-app
  name: my-app-service
spec:
  ports:
  - name: http-port
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: my-app
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-app
spec:
  replicas: 2
...
..
```

```
➜ my-app$ kustomize build | kubectl apply -f -
serviceaccount/release-name-my-app created
service/release-name-my-app created
pod/release-name-my-app-test-connection created
deployment.apps/release-name-my-app created

➜ my-app$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
my-app-5dfb54954b-b7rvt     1/1     Running   0          10s
my-app-5dfb54954b-n6752     1/1     Running   0          13s
```

Kustomize (4/4) - Phase 별로 구성

```
➜ k8s-test$ tree my-app
my-app
├── base
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── kustomization.yaml
│   ├── service.yaml
│   └── secrets.yaml
└── overlays
    ...
    ├── sandbox
    │   ├── kustomization.yaml
    │   ├── patch-resources.yaml
    │   └── patch-ingress.yaml
    ├── cbt
    │   ├── kustomization.yaml
    │   ├── patch-resources.yaml
    │   └── patch-ingress.yaml
    └── production
        ├── kustomization.yaml
        ├── patch-environments.yaml
        ├── patch-resources.yaml
        └── patch-ingress.yaml

6 directories, 14 files
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

commonLabels:
  app: my-app

resources:
- secrets.yaml
- deployment.yaml
- service.yaml
- ingress.yaml

images:
- name: sangwonl/my-app
  newTag: latest
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base

patchesStrategicMerge:
- patch-resources.yaml
- patch-ingress.yaml

images:
- name: sangwonl/my-app
  newTag: 1.51.3
```

Kustomize (4/4) - Phase 별로 구성

```
➜ k8s-test$ tree my-app
my-app
├── base
│   ├── deployment.yaml
│   ├── ingress.yaml
│   ├── kustomization.yaml
│   ├── service.yaml
│   └── secrets.yaml
└── overlays
    ...
    ├── sandbox
    │   ├── kustomization.yaml
    │   ├── patch-resources.yaml
    │   └── patch-ingress.yaml
    ├── cbt
    │   ├── kustomization.yaml
    │   ├── patch-resources.yaml
    │   └── patch-ingress.yaml
    └── production
        ├── kustomization.yaml
        ├── patch-environments.yaml
        ├── patch-resources.yaml
        └── patch-ingress.yaml

6 directories, 14 files
```

```
➜ my-app$ cd overlays/sandbox
➜ my-app$ kustomize build

or

➜ my-app$ kustomize build overlays/sandbox
```

3. 활용 패턴

클러스터 분리 및 이름 규칙

- 서비스 성격에 따라 여러 클러스터로 분리

my-data-02 → 데이터 파이프라인 및 데이터 집계/분석 관련 앱들

my-dev-04 → 개발용 클러스터

my-prod-02 → 운영용 클러스터

my-prod-03 → 운영용 신규 클러스터

my-devops-01 → Jenkins, ArgoCD 등 DevOps 관련 앱들

my-pm-01 → PM worker들로 이루어진 클러스터, PM에 띄워야할 앱을 위해

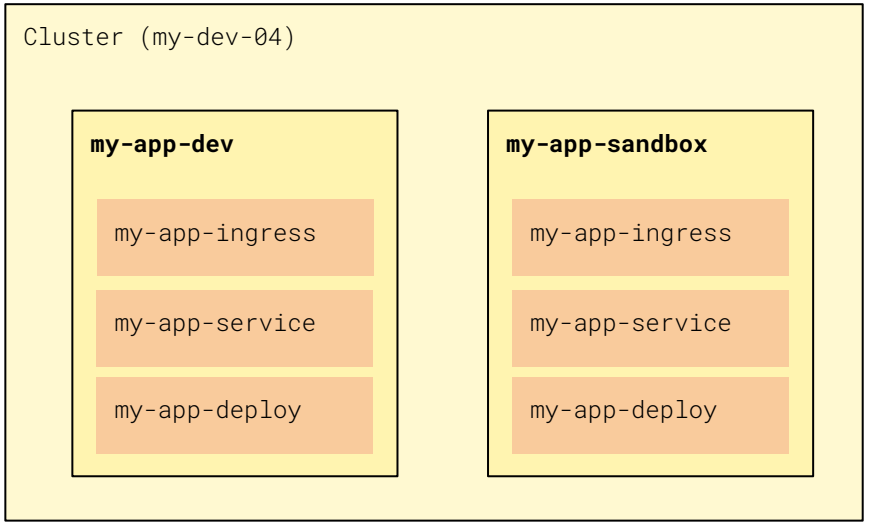
- 클러스터 이름 규칙

<org>-<purpose>-<index> (index는 클러스터 이전마다 증가)

네임스페이스 용도 및 이름 규칙

- 네임스페이스를 쓰면?

* k8s 오브젝트들의 경계를 칠 수 있음
* 네임스페이스가 다르면 오브젝트 이름이 동일해도 됨
* 보통 앱 설정은 Phase 별로 거의 동일
  → 네임스페이스로 Phase를 구분 가능



- 네임스페이스 이름 규칙

<app-name>-<phase>

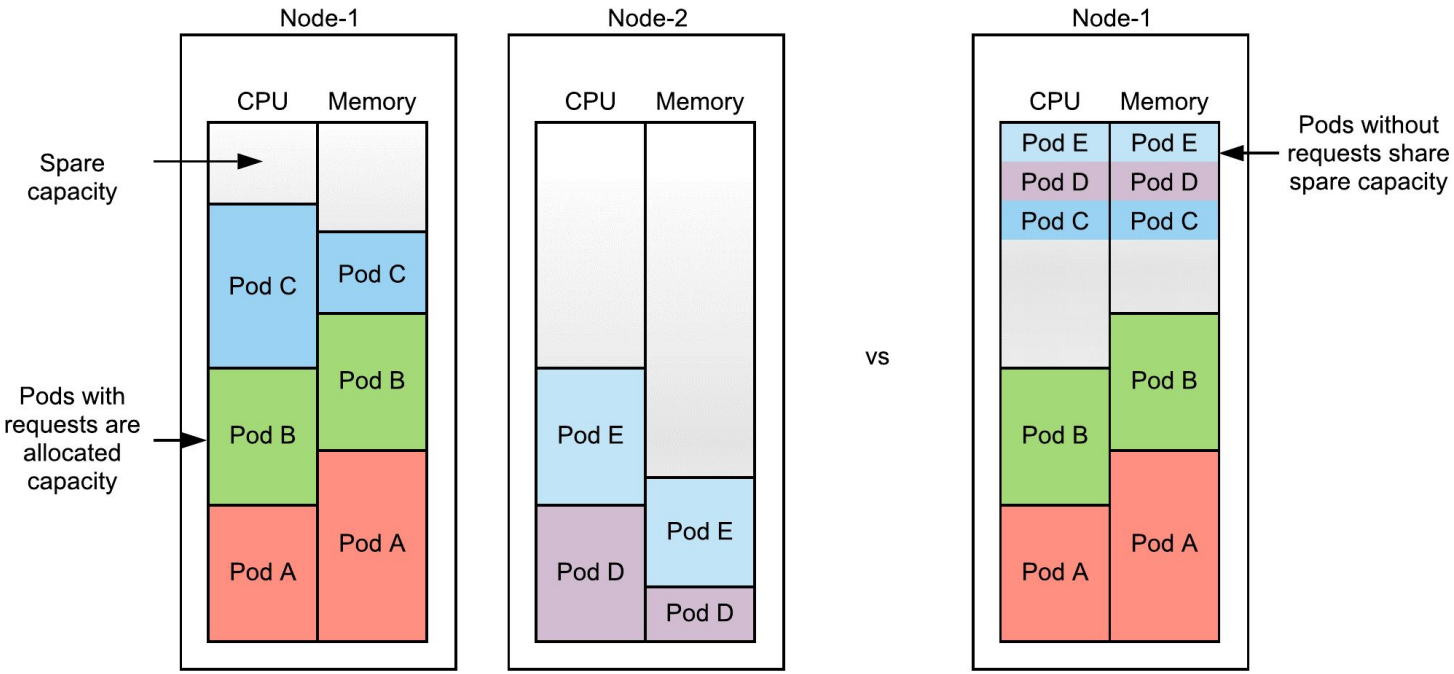
* **phase** → dev, sandbox, cbt, production
* **dev** cluster → dev / sandbox
  prod cluster → cbt / production

<cluster>-<common>

* 클러스터 공통인 앱들, default를 써도 됨

ex) my-app-sandbox, my-app-production, jenkins-common

배치 / 스케쥴링 (1/3) - Pod Resource



배치 / 스케쥴링 (1/3) - Pod Resource

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        resources:
          requests:
            cpu: 2500m
            memory: 2Gi
          limits:
            cpu: 4
            memory: 4Gi
```

배치 / 스케쥴링 (1/3) - Pod Resource

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        resources:
          requests:
            cpu: 2500m
            memory: 2Gi
          limits:
            cpu: 4
            memory: 4Gi
```

2Gi 메모리와 CPU 2.5 코어 만큼의 여유가 되는 노드가 있다면 해당 노드로 스케쥴링 가능

1 CPU = 1000m (Millicores)

배치 / 스케쥴링 (1/3) - Pod Resource

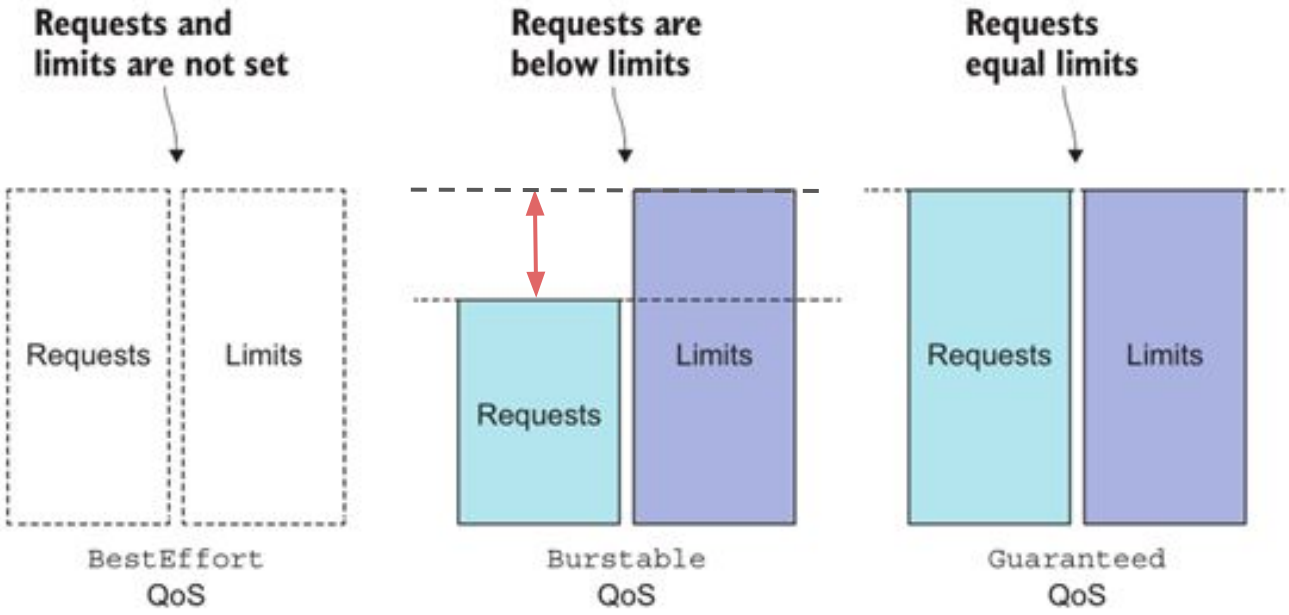
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        resources:
          requests:
            cpu: 2500m
            memory: 2Gi
          limits:
            cpu: 4
            memory: 4Gi
```

2Gi 메모리와 CPU 2.5 코어 만큼의 여유가 되는 노드가 있다면 해당 노드로 스케쥴링 가능

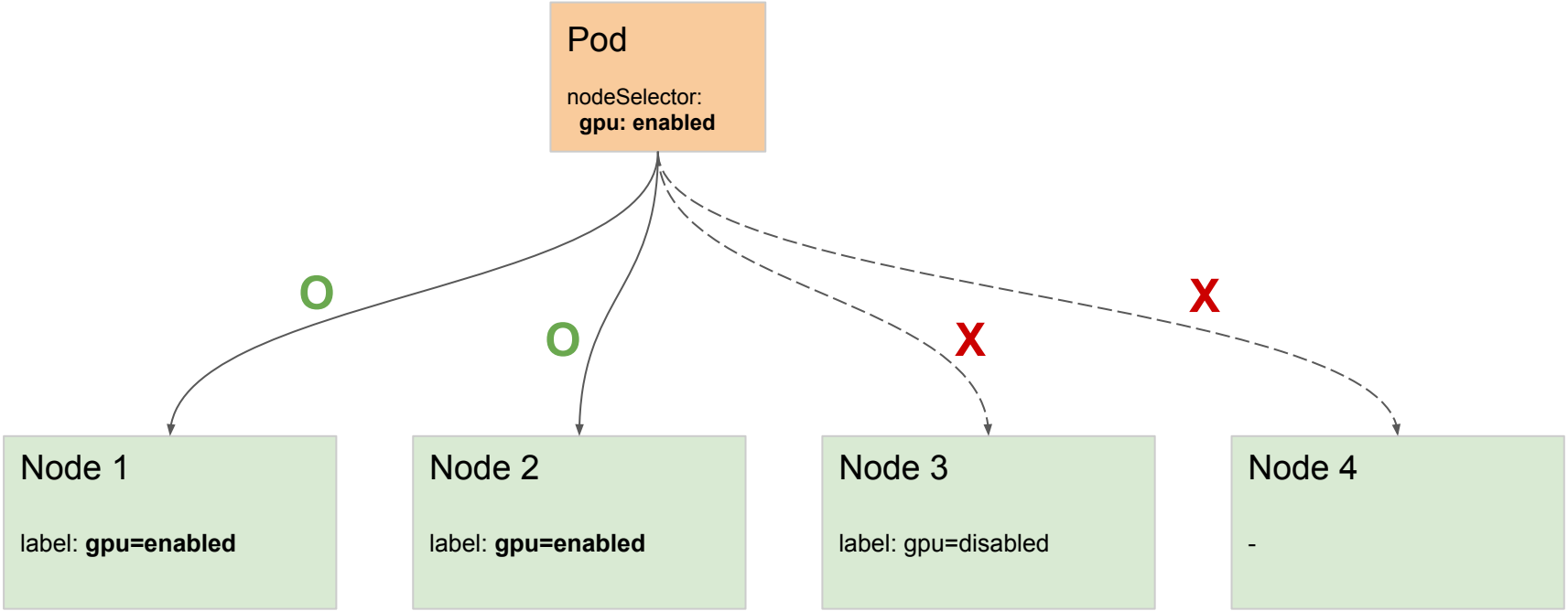
Pod의 CPU 사용량이 4 코어를 넘어가면 사용량이 제한(Throttling)되기 시작. 한편, 메모리 사용량이 4Gi를 넘어가게 되면 Killed 될 수 있음.

1 CPU = 1000m (Millicores)

배치 / 스케쥴링 (1/3) - Pod Resource



배치 / 스케쥴링 (2/3) - Node Selector



배치 / 스케쥴링 (2/3) - Node Selector

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
      nodeSelector:
        gpu: enabled
```

Pod

nodeSelector:
gpu: enabled

O

O

X

Node 1

label: gpu=enabled

Node 2

label: gpu=enabled

Node 3

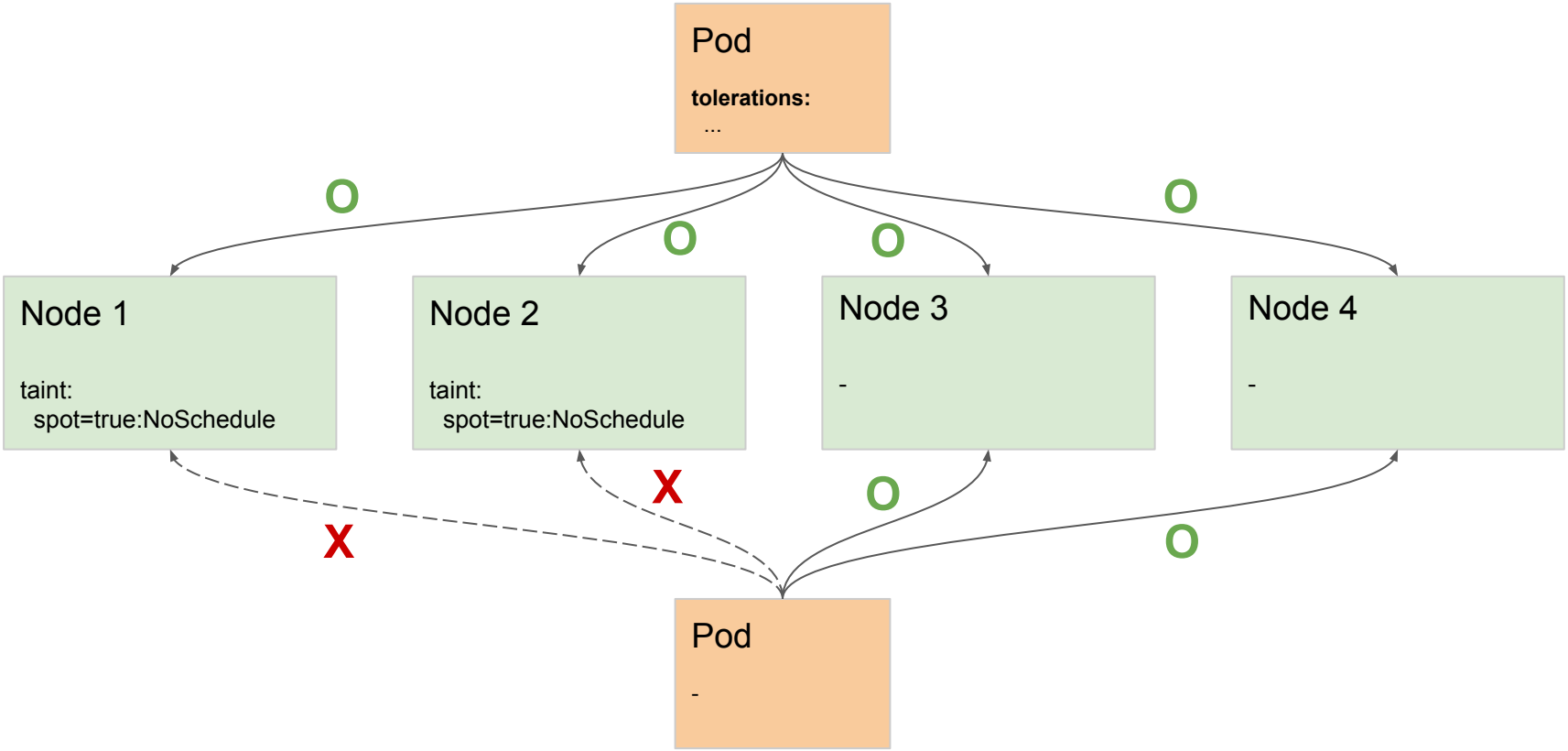
label: gpu=disabled

Node 4

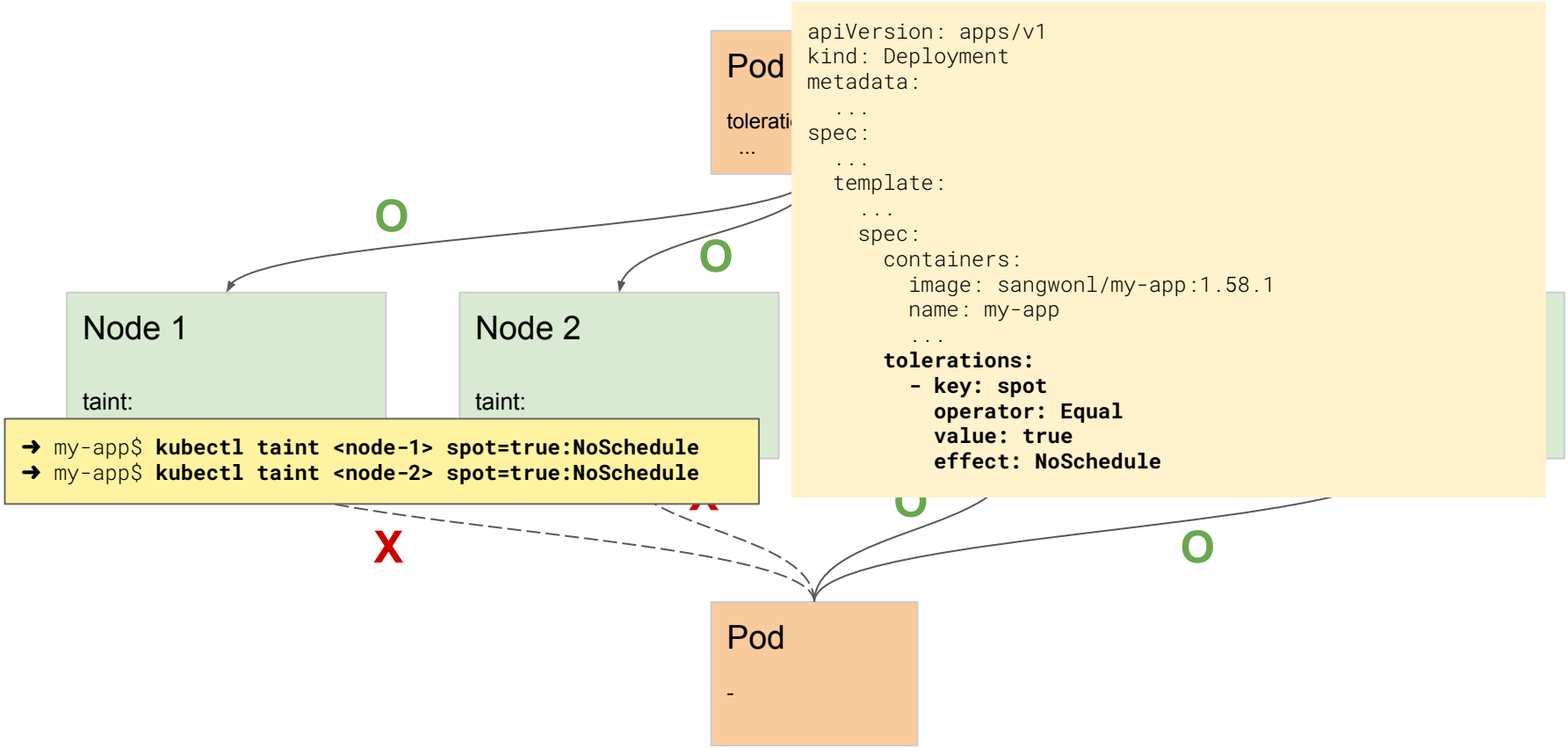
-

```
➜ my-app$ kubectl label <node-1> gpu=enabled
➜ my-app$ kubectl label <node-2> gpu=enabled
```

배치 / 스케쥴링 (3/3) - Taint / Toleration

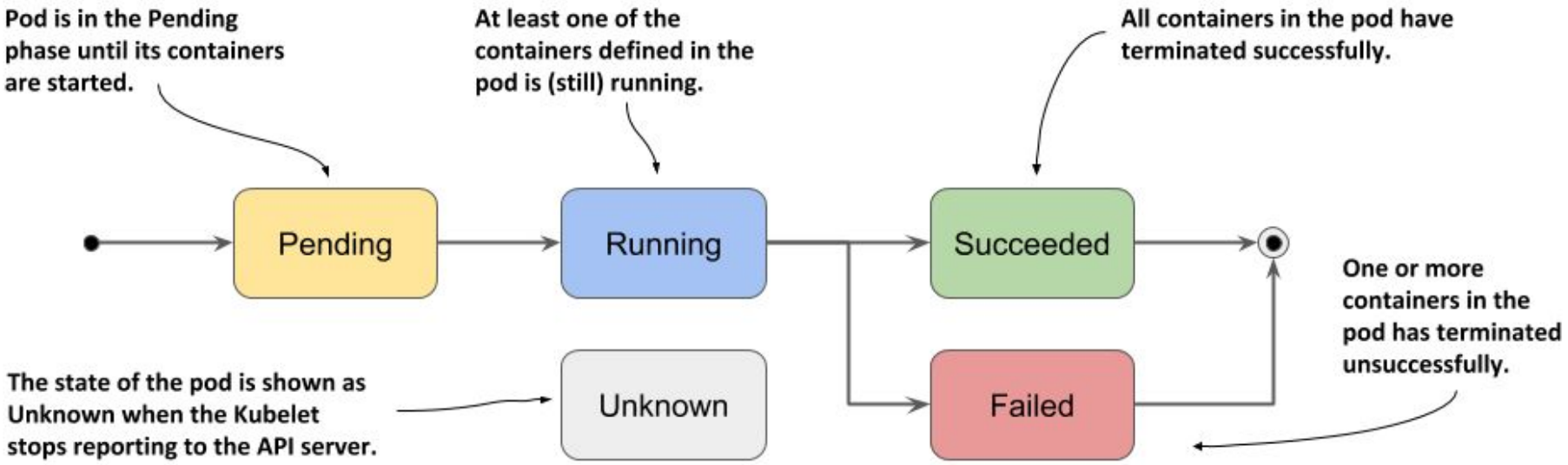


배치 / 스케쥴링 (3/3) - Taint / Toleration

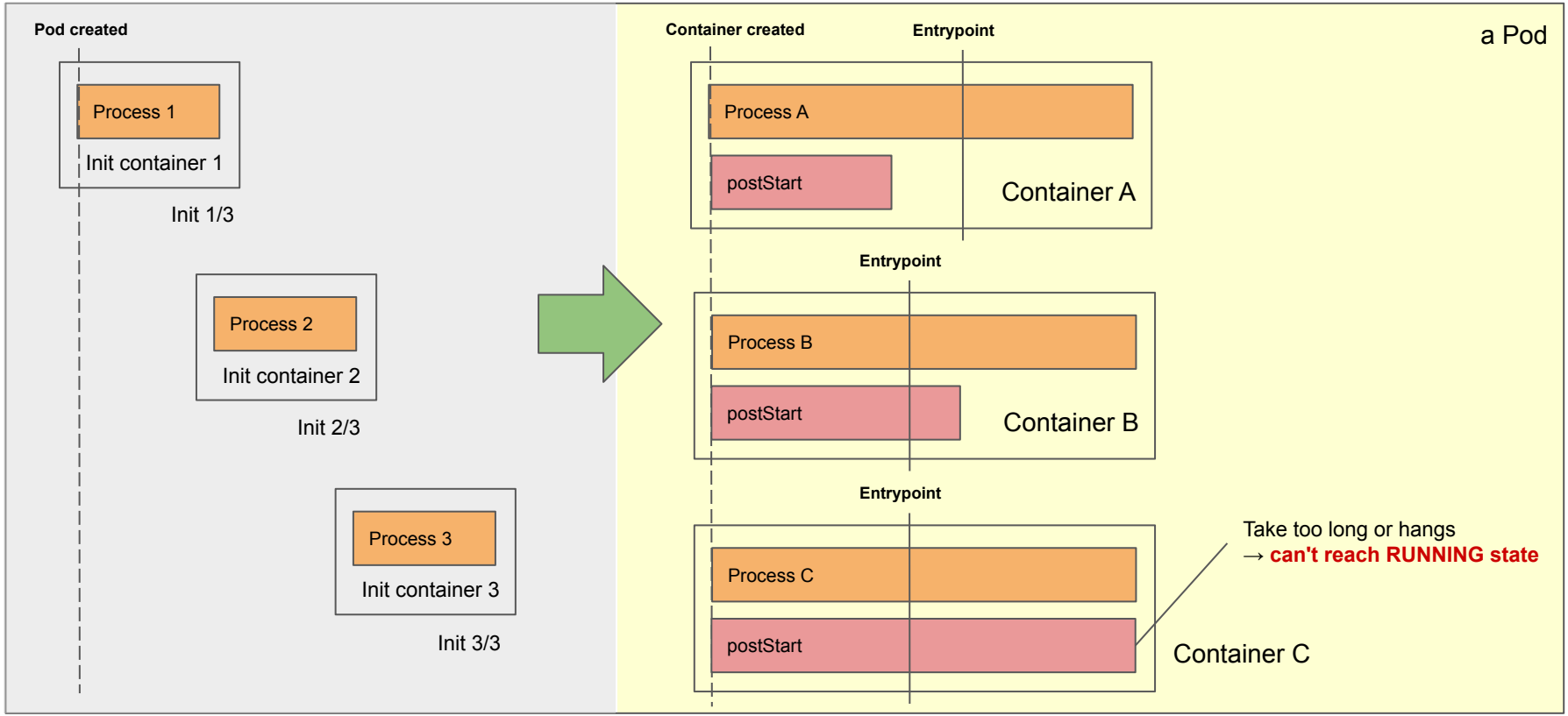



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
      tolerations:
        - key: spot
          operator: Equal
          value: true
          effect: NoSchedule
```

라이프사이클 (1/4) - Pod 상태

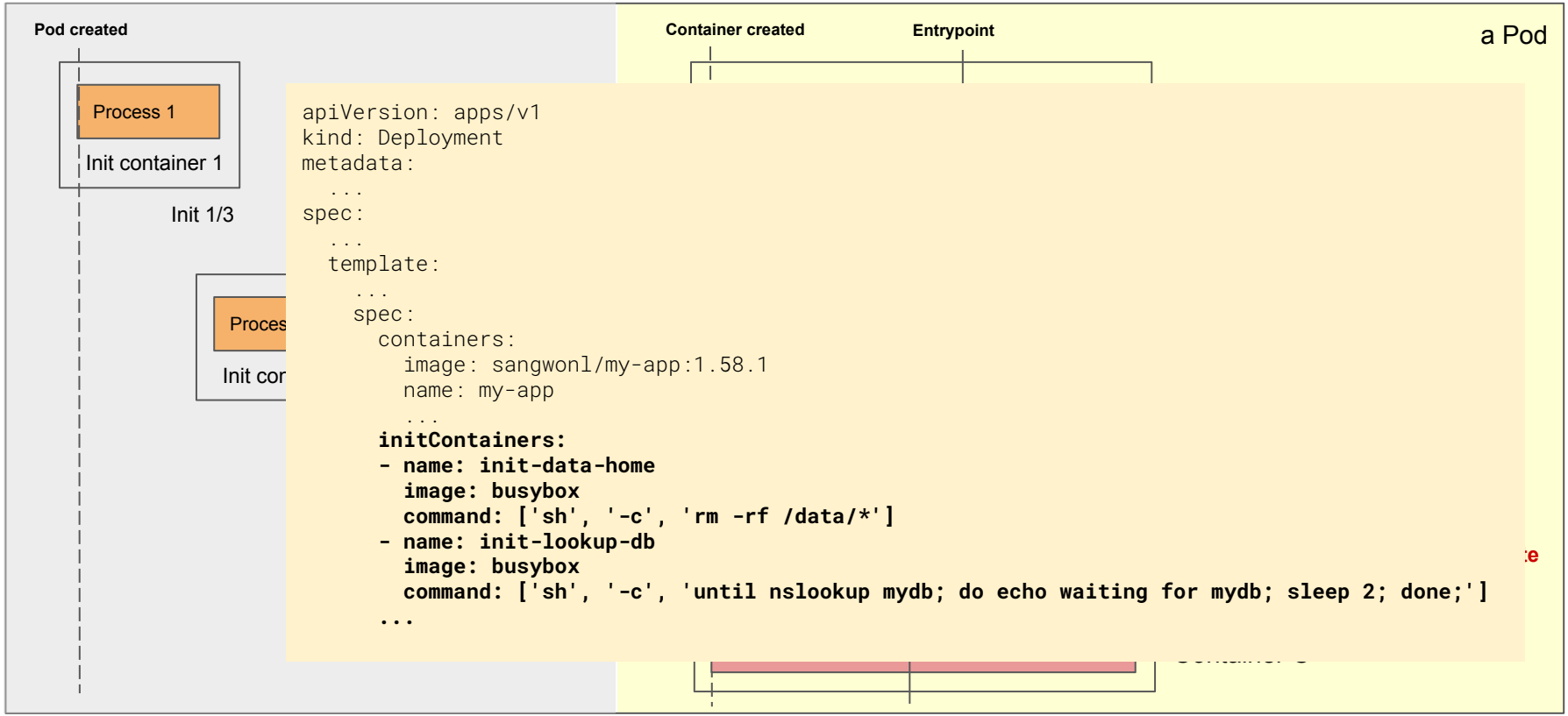


라이프사이클 (2/4) - Hook (postStart)



https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#hook-details

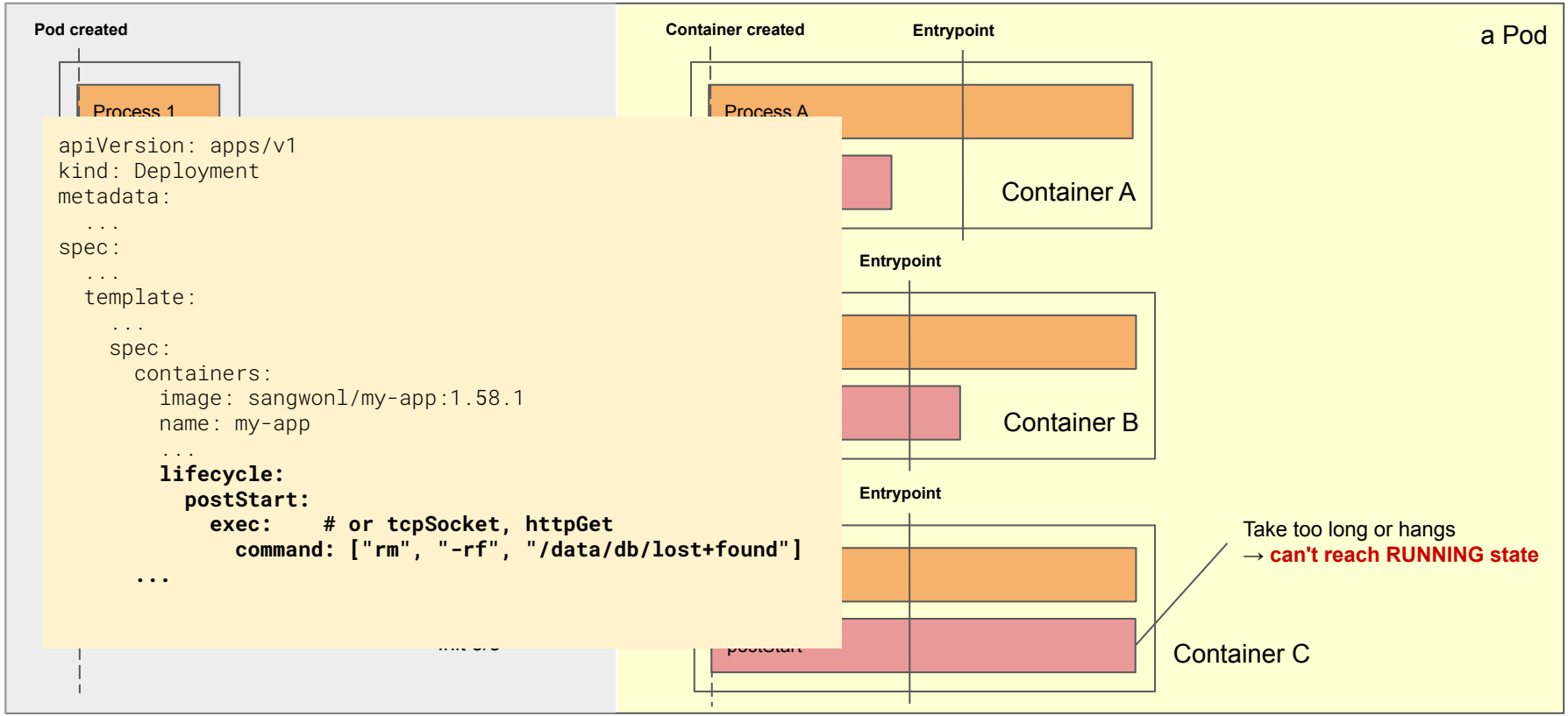
라이프사이클 (2/4) - Hook (postStart)




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
      initContainers:
      - name: init-data-home
        image: busybox
        command: ['sh', '-c', 'rm -rf /data/*']
      - name: init-lookup-db
        image: busybox
        command: ['sh', '-c', 'until nslookup mydb; do echo waiting for mydb; sleep 2; done;']
      ...
```

https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#hook-details

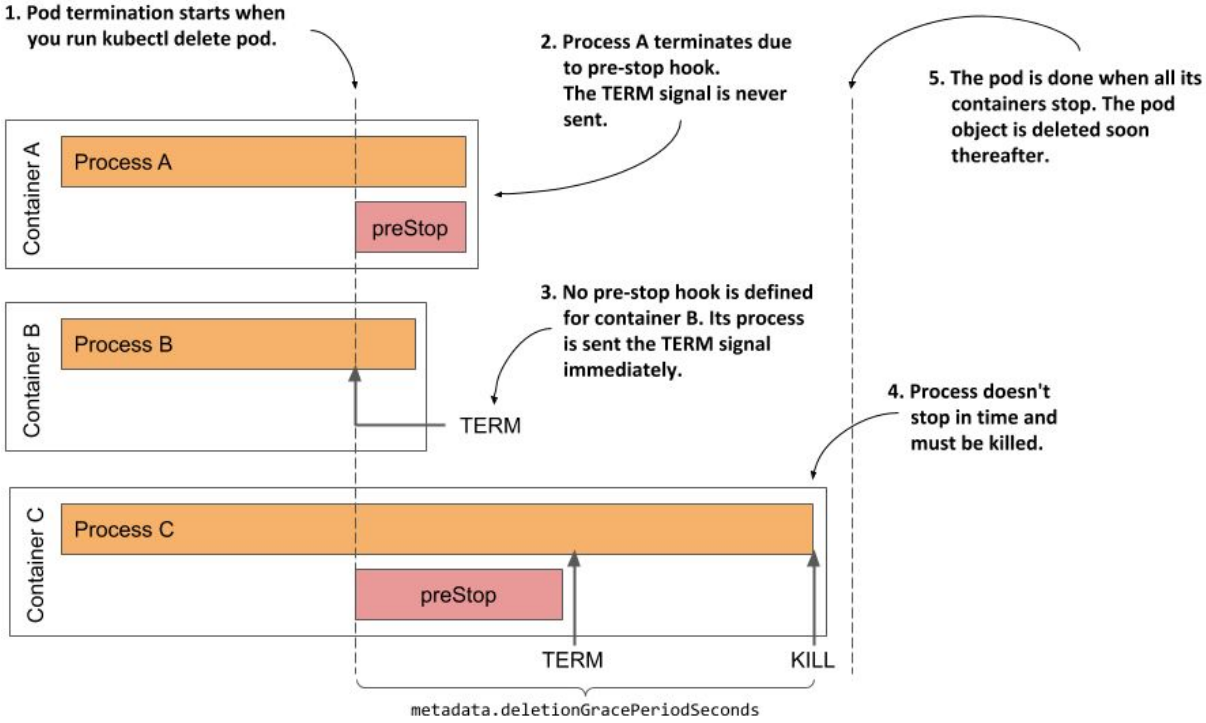
라이프사이클 (2/4) - Hook (postStart)



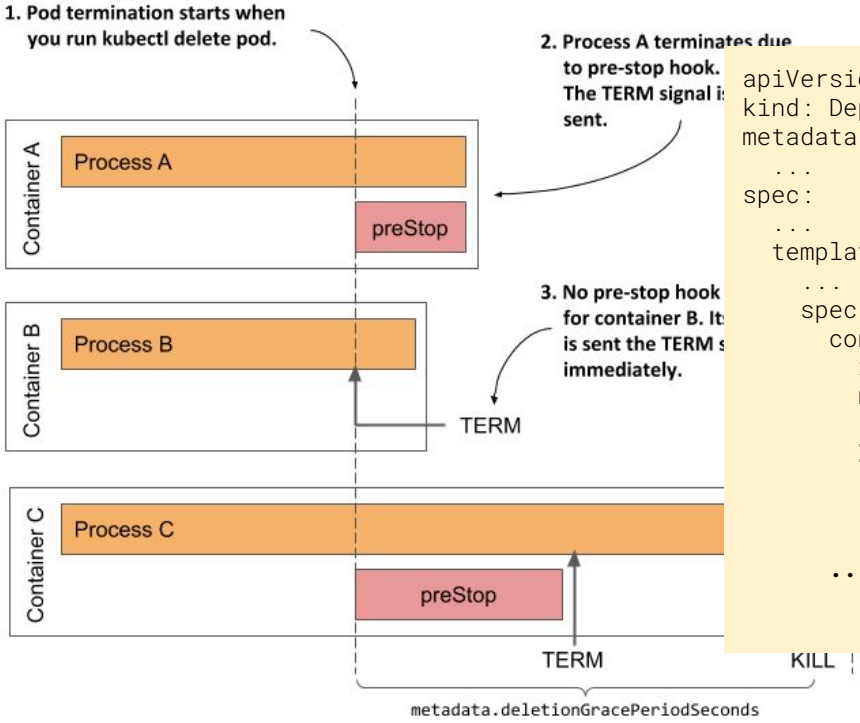

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        lifecycle:
          postStart:
            exec:    # or tcpSocket, httpGet
              command: ["rm", "-rf", "/data/db/lost+found"]
      ...
```

https://kubernetes.io/docs/concepts/containers/container-lifecycle-hooks/#hook-details

라이프사이클 (2/4) - Hook (preStop)



라이프사이클 (2/4) - Hook (preStop)




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        lifecycle:
          preStop:
            exec:    # or tcpSocket, httpGet
              command: ["./stop-consuming.sh"]
      ...
```

라이프사이클 (3/4) - Readiness Probe (HTTP)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  ...
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: sangwonl/my-app:latest
        ports:
        - containerPort: 9090
        readinessProbe:
          httpGet:
            port: 9090
            path: /healthcheck
          initialDelaySeconds: 60
          periodSeconds: 10
          timeoutSeconds: 1
          successThreshold: 1
          failureThreshold: 3
```

GET http://localhost:9090/healthcheck

initialDelaySeconds: 60
컨테이너가 시작된 후 60초 동안은 probe ping 없이 기다림

periodSeconds: 10
얼마나 주기적으로 probe 할 것인지, 기본값 10초

timeoutSeconds: 1
probe 응답의 타임아웃, probe가 1초 넘어가면 실패로 간주, 기본값 1초

successThreshold: 1
probe가 최소 몇 번 성공해야 readiness를 성공이라고 판단할지, 기본값 1

failureThreshold: 3
probe가 최소 몇 번 실패해야 readiness를 실패라고 판단할지, 기본값 3

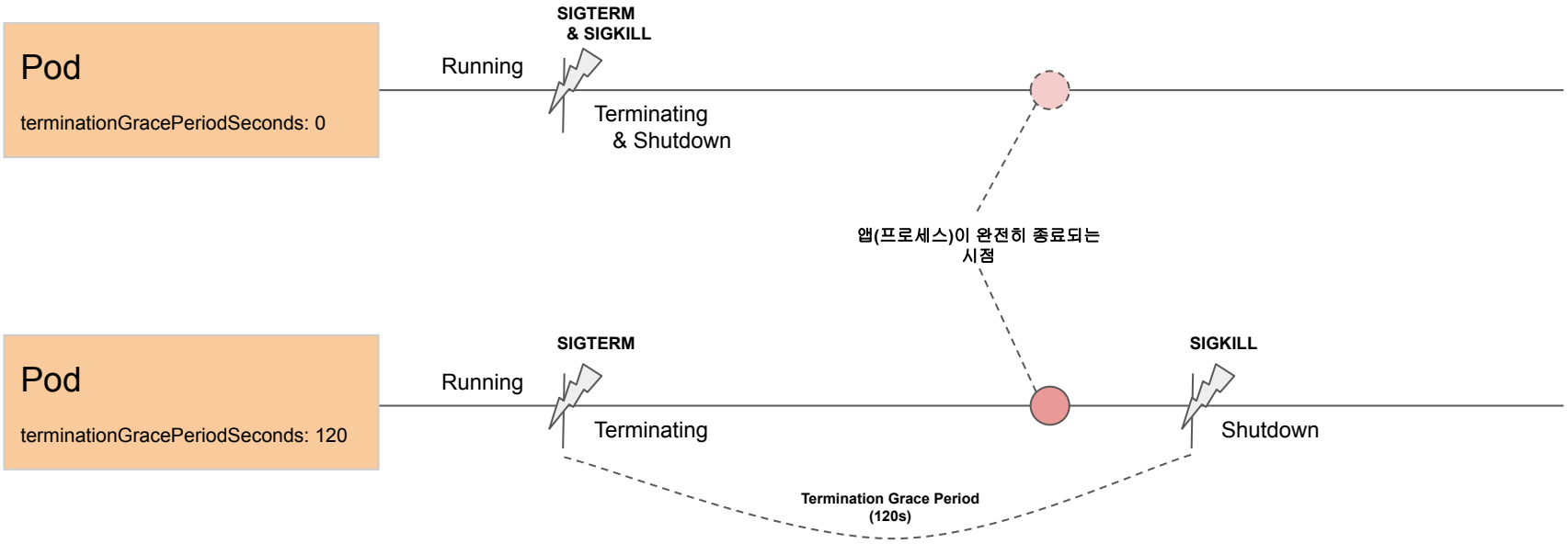
라이프사이클 (3/4) - Readiness Probe (gRPC)

```yaml
apiVersion: apps/v1
kind: Deployr
metadata:
  name: my-a
spec:
  ...
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: sangwonl/my-app:latest
        ports:
        - containerPort: 9090
        readinessProbe:
          exec:
            command:                # 커맨드 실행 방식
            - grpc_health_probe     # 도커이미지에 미리 설치해둔 실행 파일
            - -addr=:9090           # 실행 파일의 인자
          initialDelaySeconds: 60
```

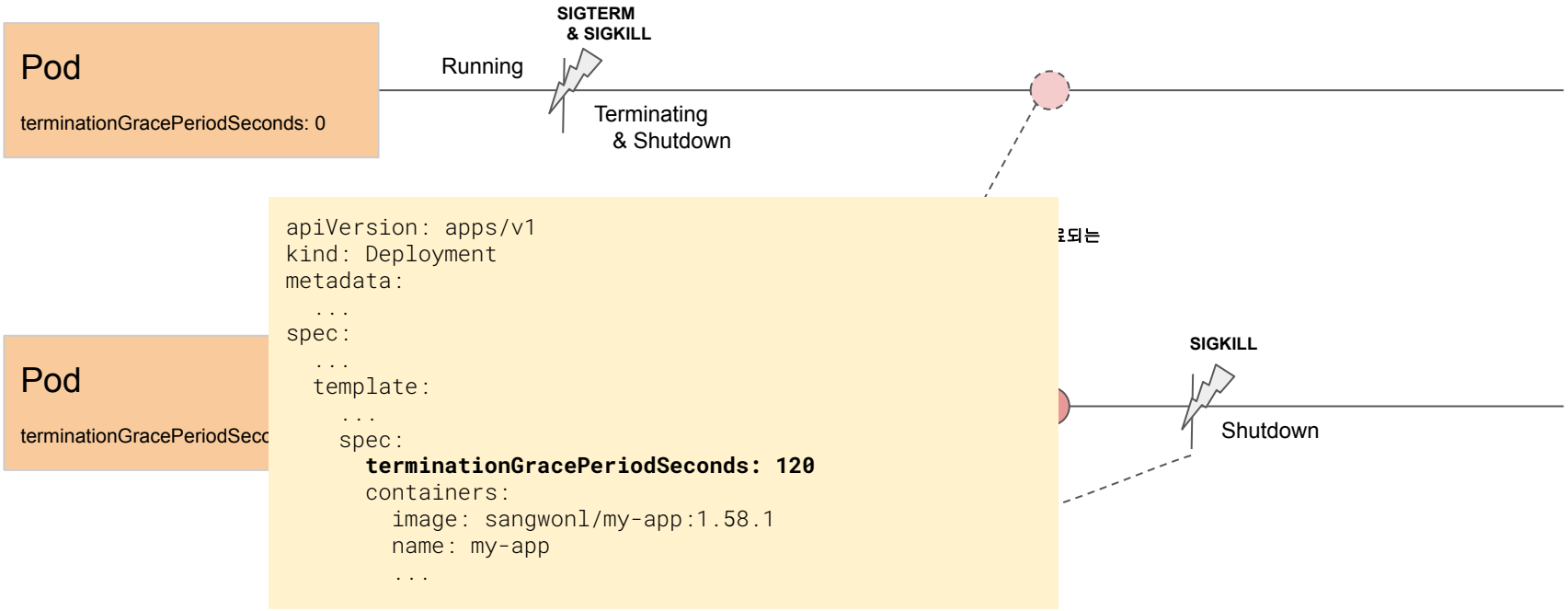
```
$ grpc_health_probe -addr=:9090
```

```
➜ my-app$ cat Dockerfile
...
...
# Install grpc_health_probe
RUN wget -qO/bin/grpc_health_probe
https://github.com/grpc-ecosystem/grpc-health-probe/releases/download/v0.3
.2/grpc_health_probe-linux-amd64 && \
  chmod +x /bin/grpc_health_probe
...
```

라이프사이클 (4/4) - Termination Grace Period

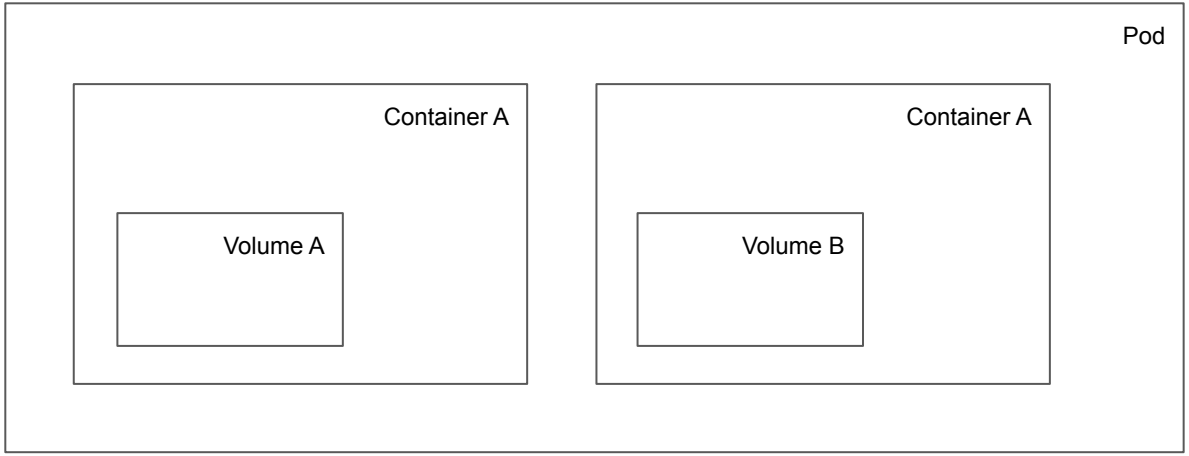


라이프사이클 (4/4) - Termination Grace Period

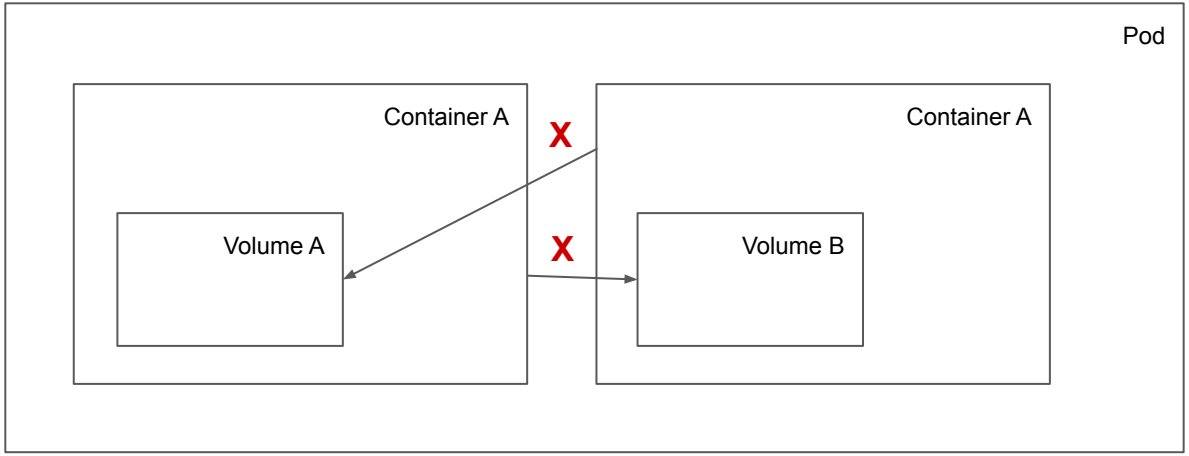



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      terminationGracePeriodSeconds: 120
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
```

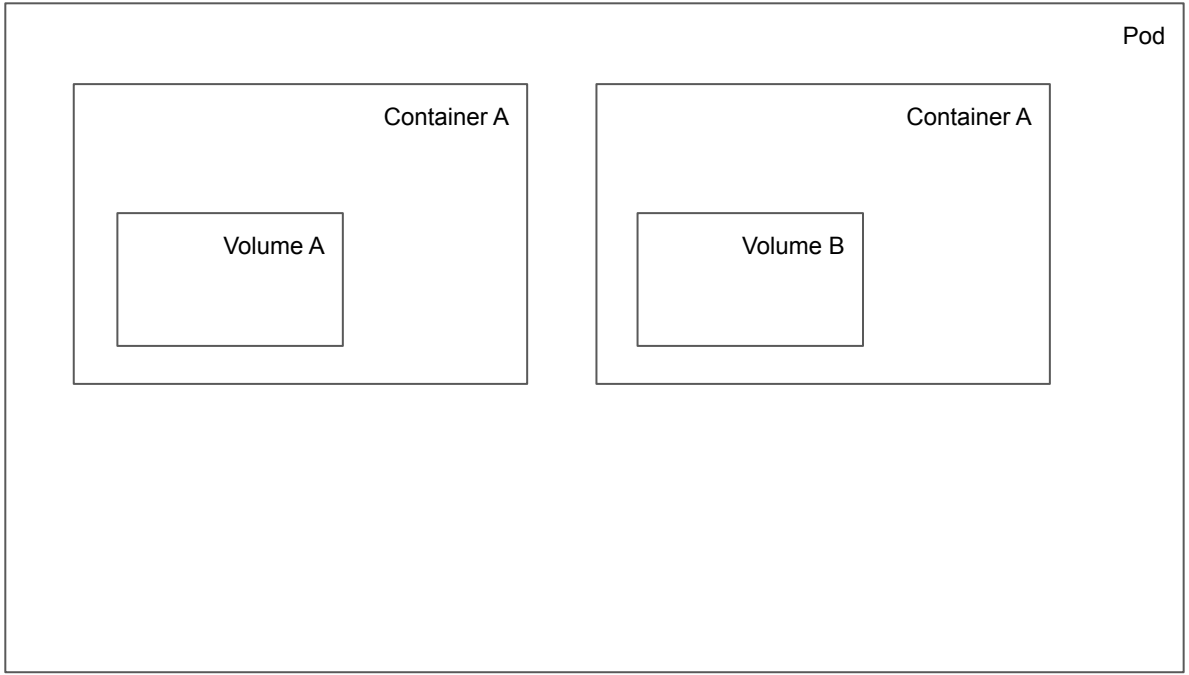
볼륨 (1/4) - 기본



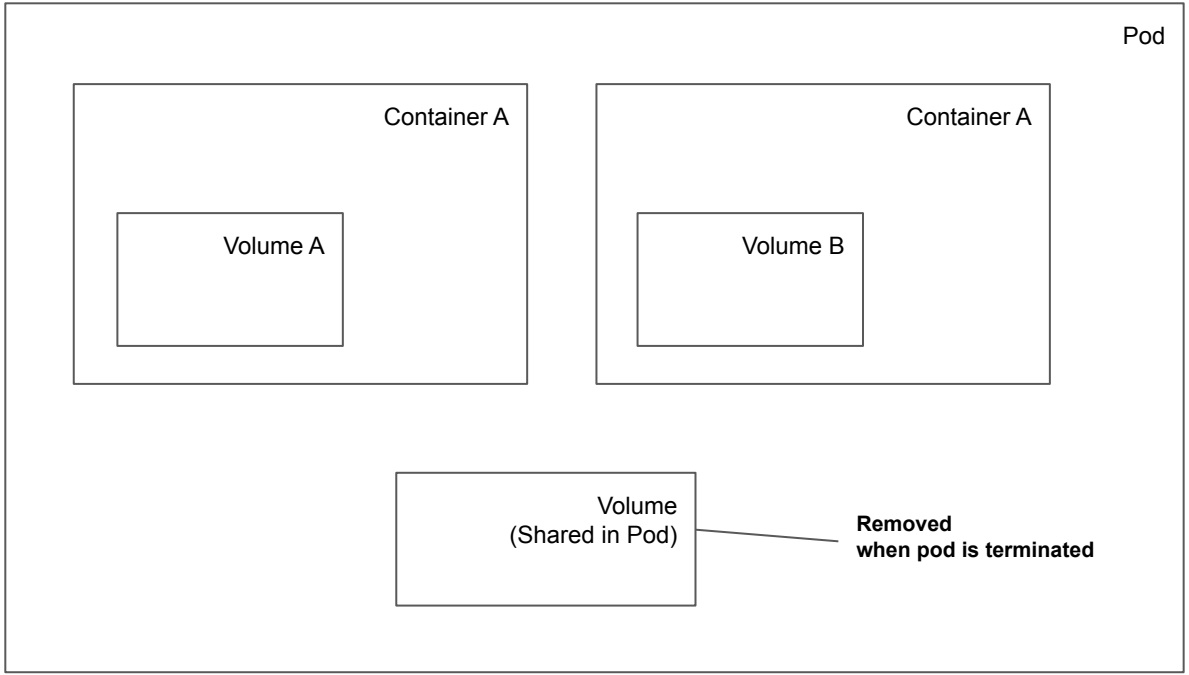
볼륨 (1/4) - 기본



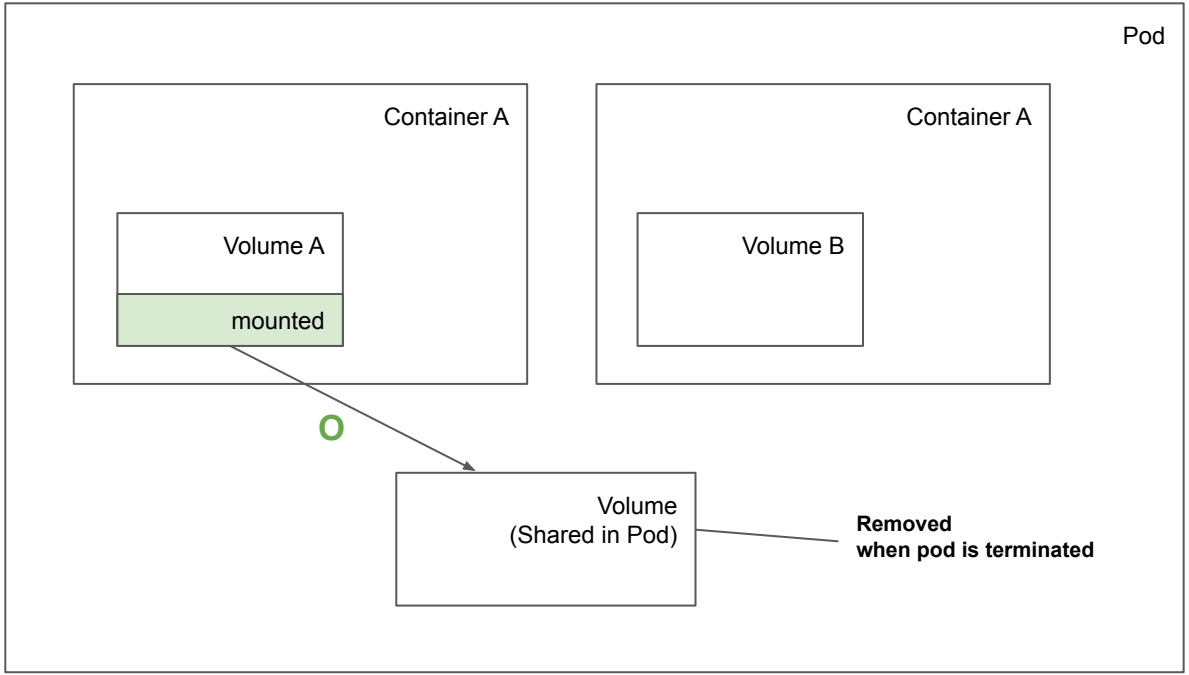
볼륨 (1/4) - Pod 스코프 임시 볼륨 (emptyDir)



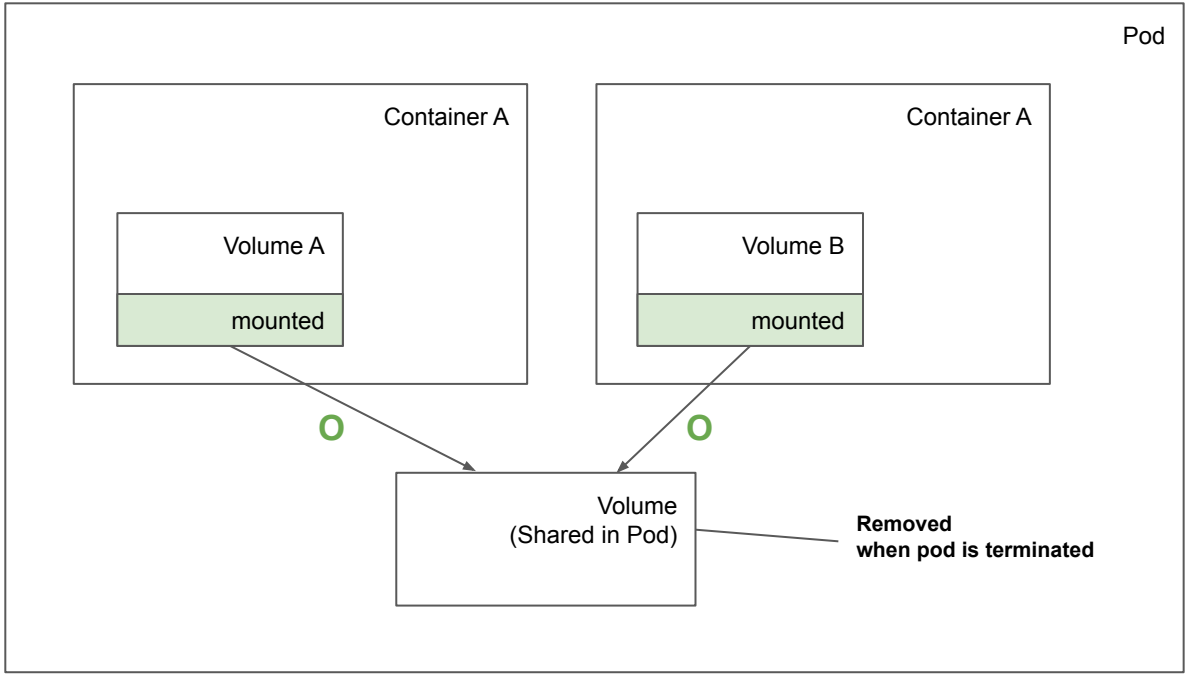
볼륨 (1/4) - Pod 스코프 임시 볼륨 (emptyDir)



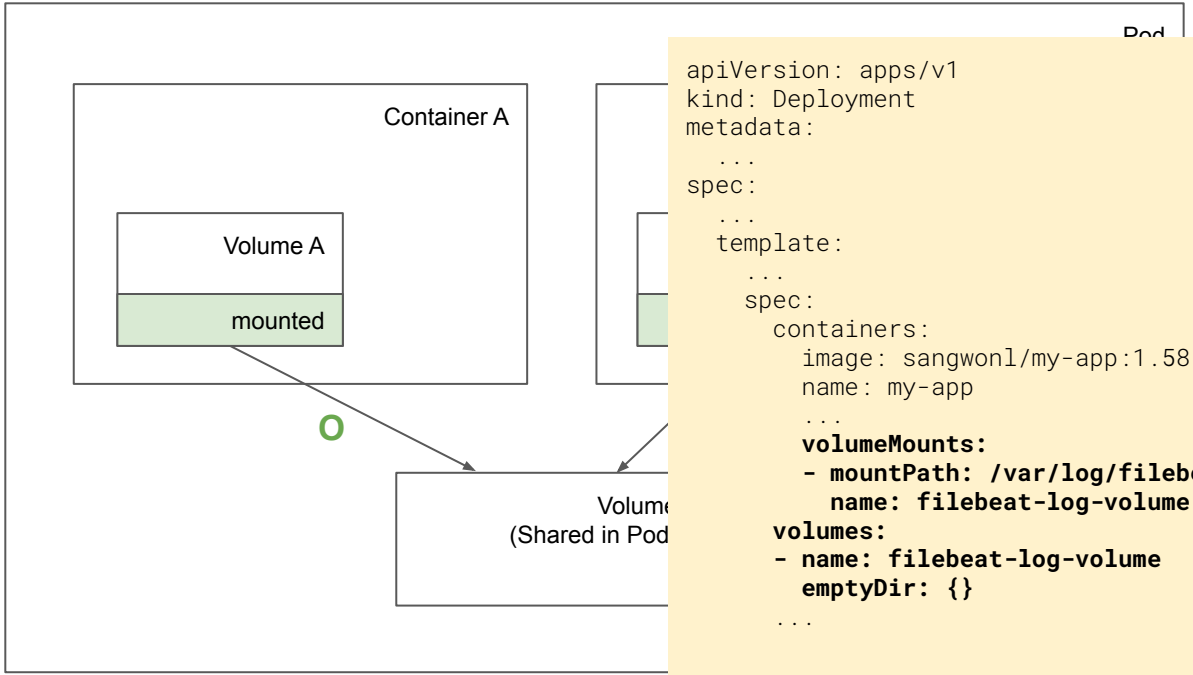
볼륨 (1/4) - Pod 스코프 임시 볼륨 (emptyDir)



볼륨 (1/4) - Pod 스코프 임시 볼륨 (emptyDir)

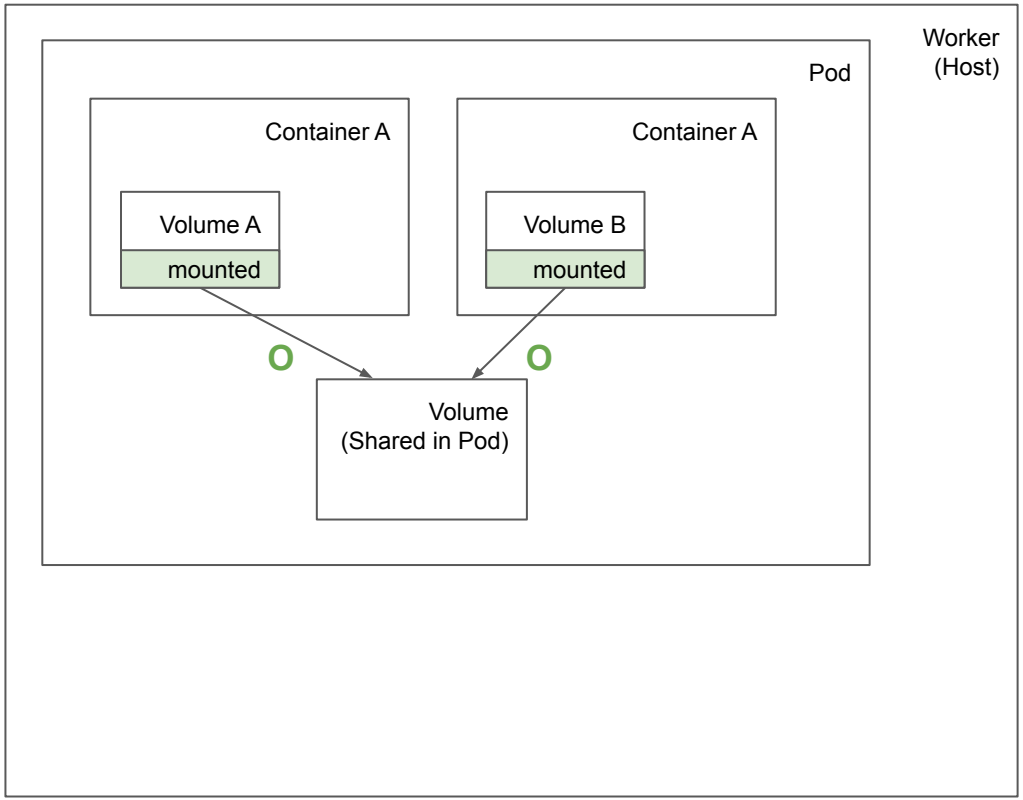


볼륨 (1/4) - Pod 스코프 임시 볼륨 (emptyDir)

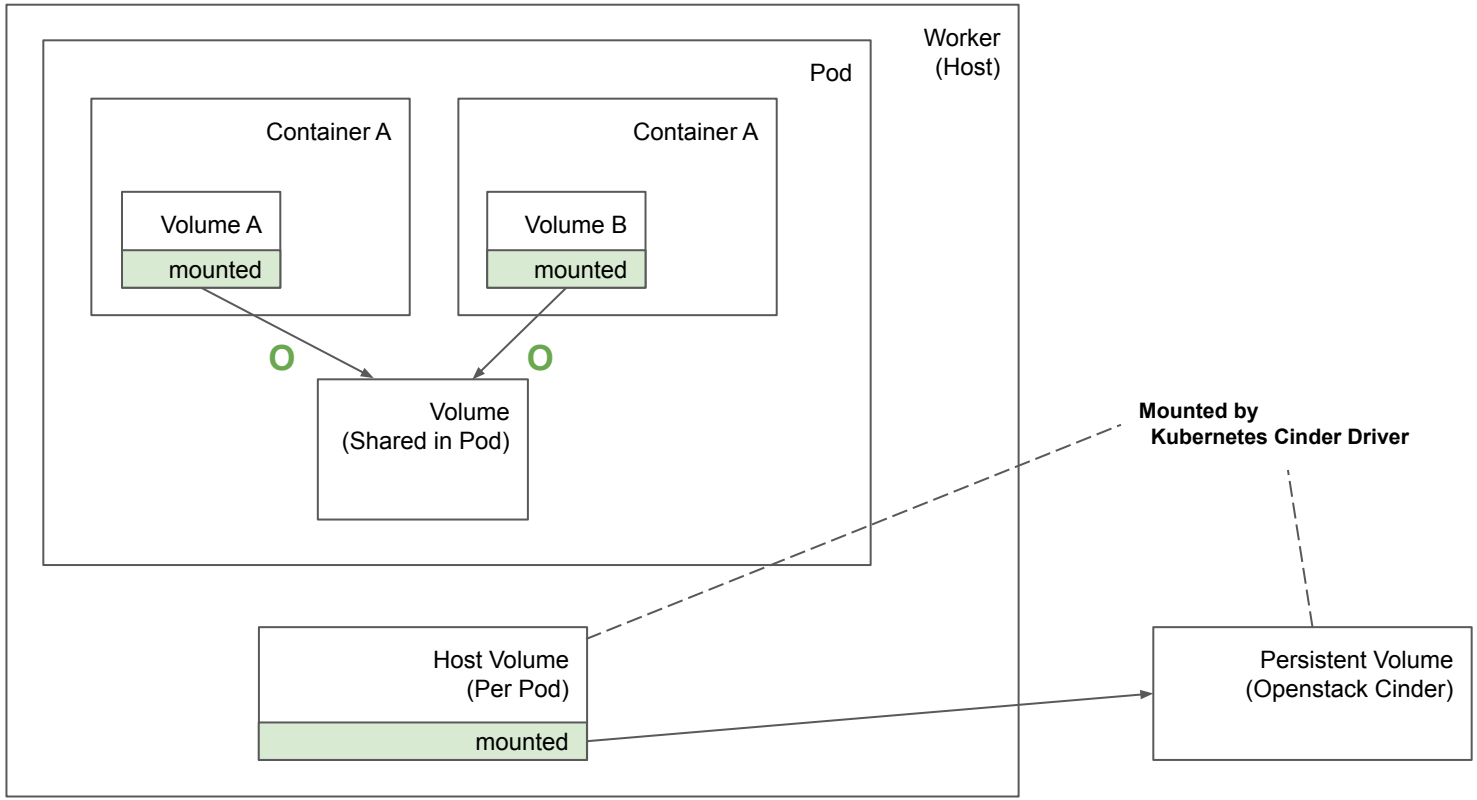



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        volumeMounts:
        - mountPath: /var/log/filebeat
          name: filebeat-log-volume
      volumes:
      - name: filebeat-log-volume
        emptyDir: {}
      ...
```

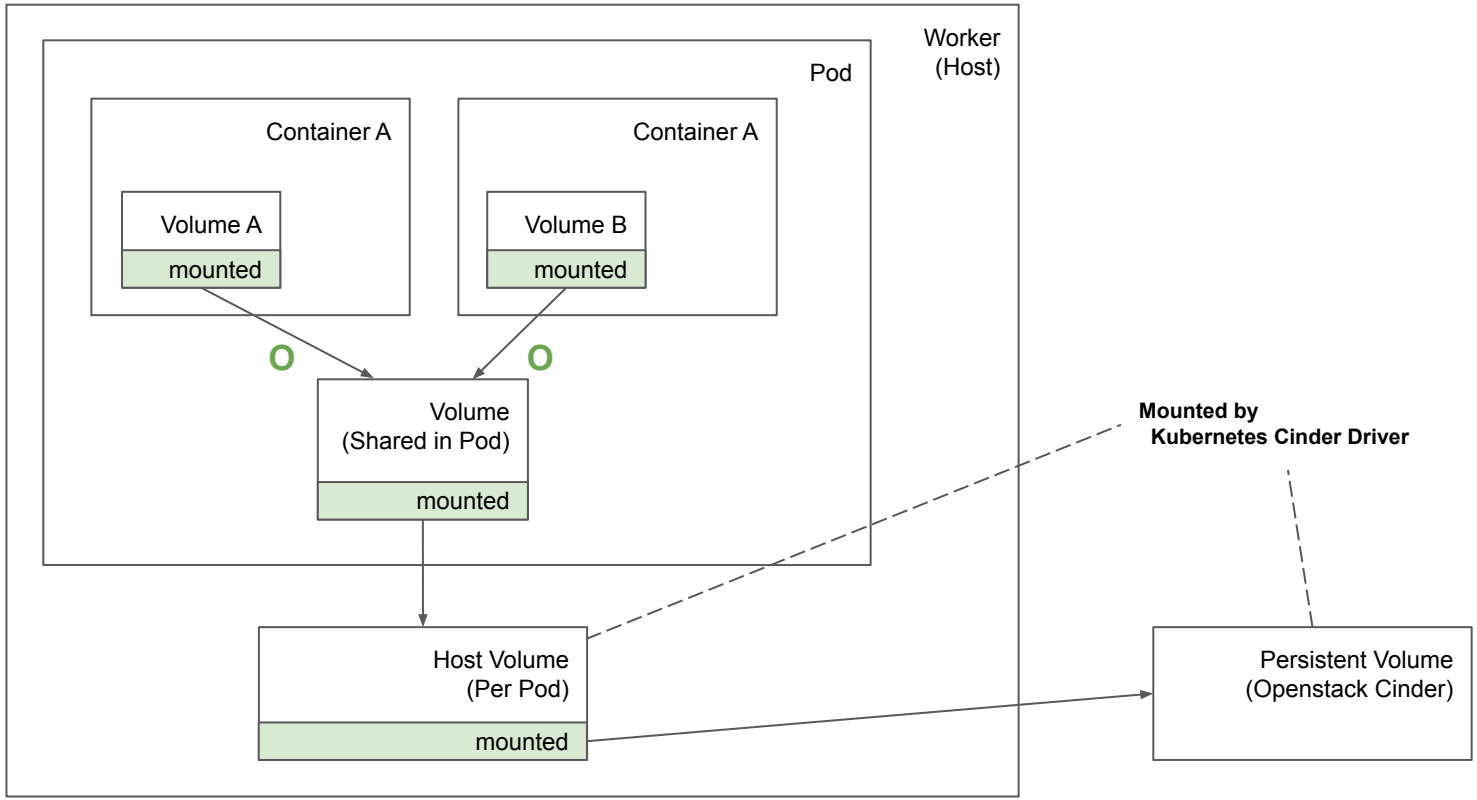
볼륨 (2/4) - Persistent Volume (ReadWriteOnce)



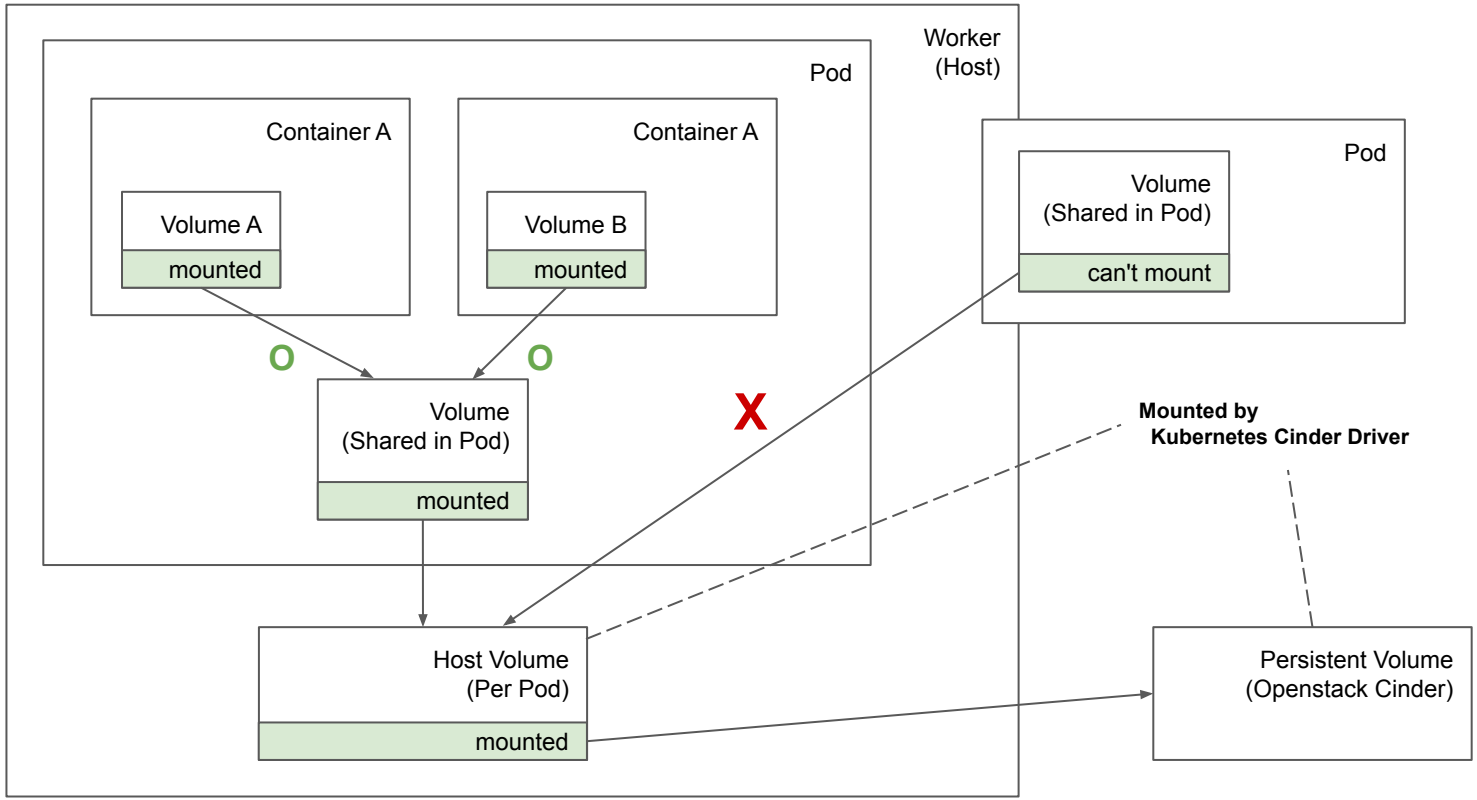
볼륨 (2/4) - Persistent Volume (ReadWriteOnce)



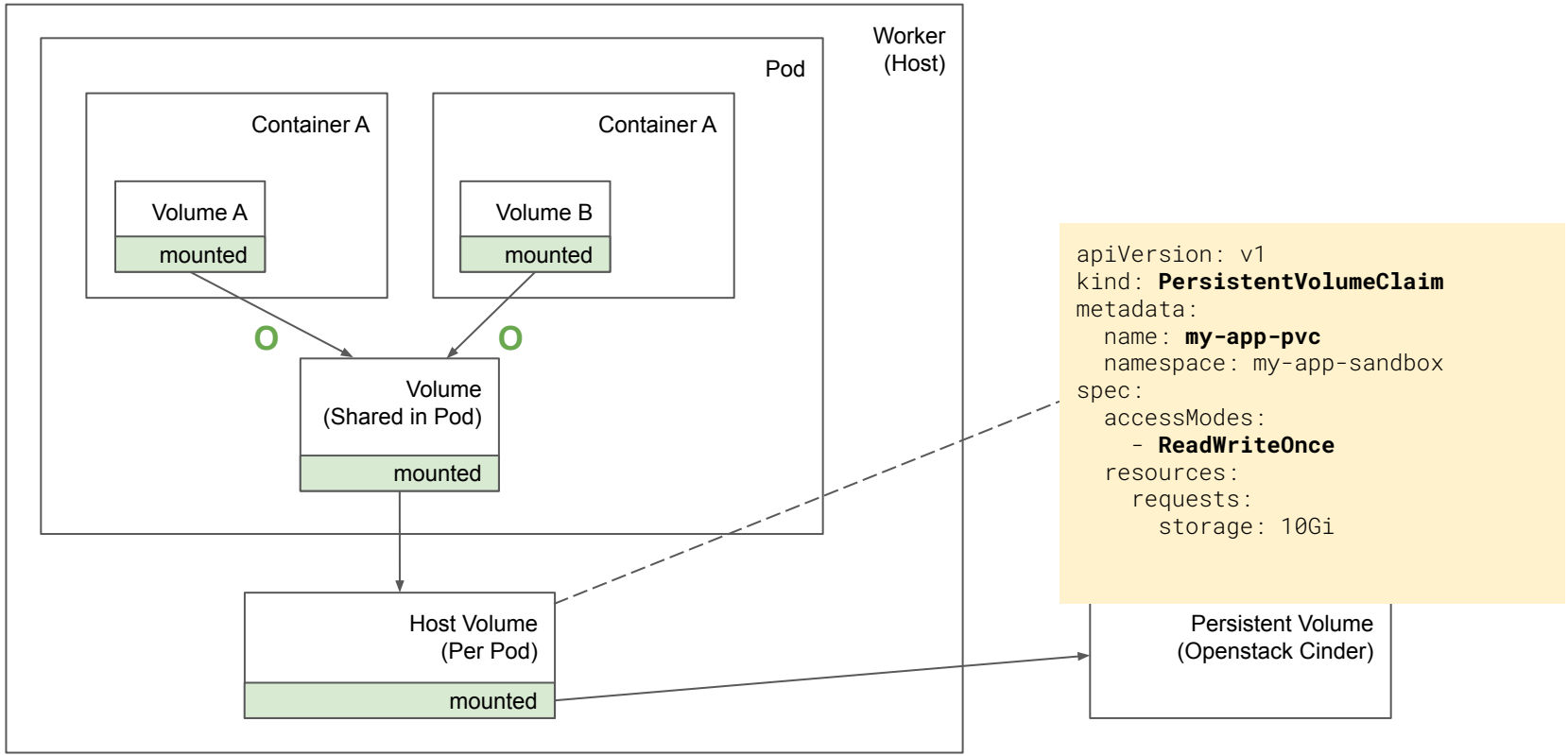
볼륨 (2/4) - Persistent Volume (ReadWriteOnce)



볼륨 (2/4) - Persistent Volume (ReadWriteOnce)

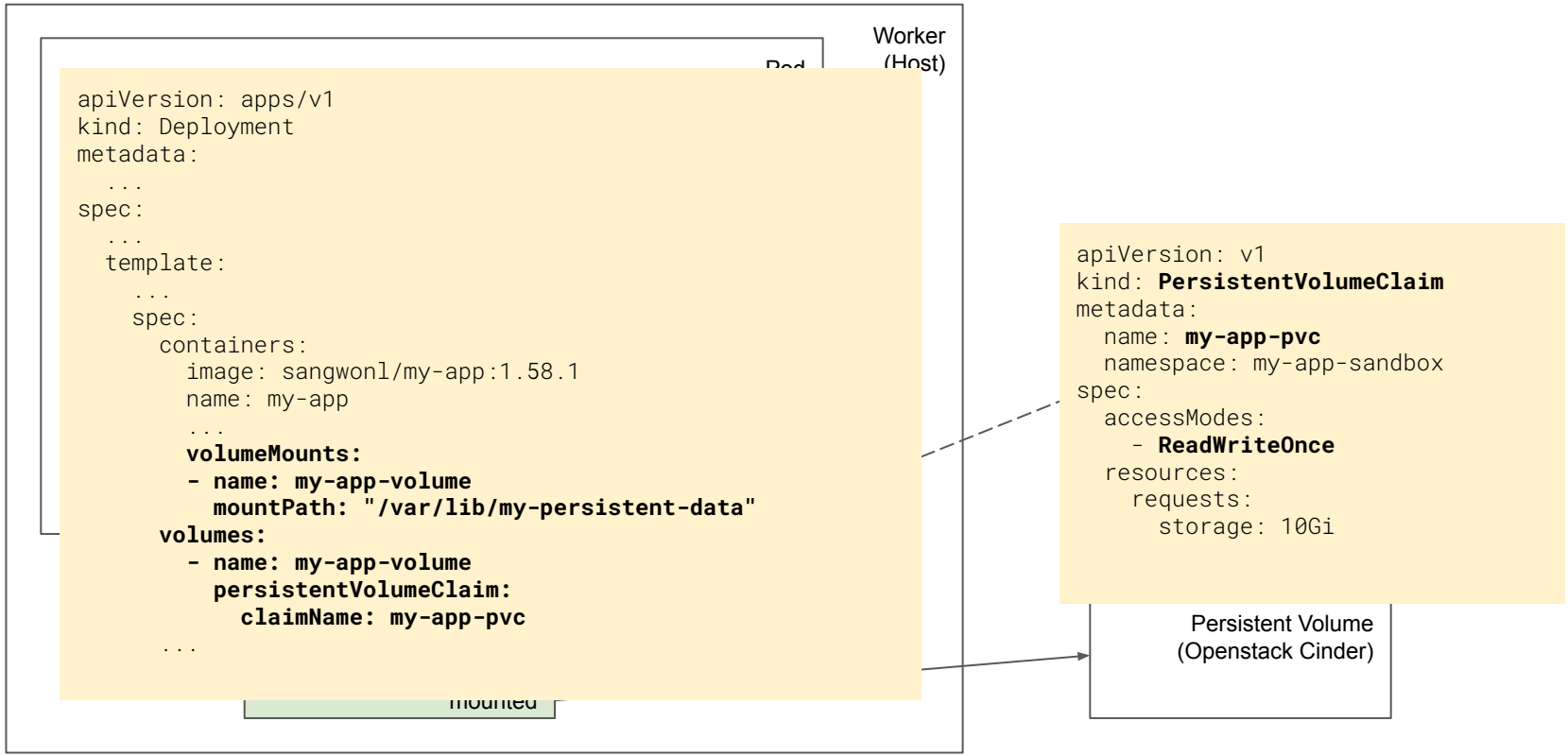


볼륨 (2/4) - Persistent Volume (ReadWriteOnce)

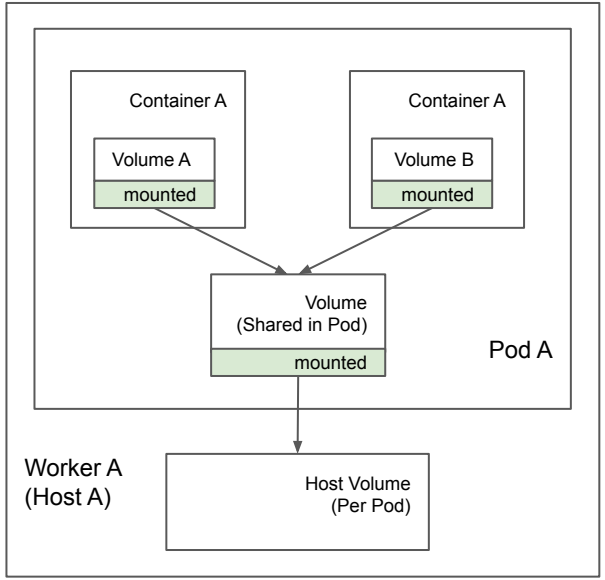


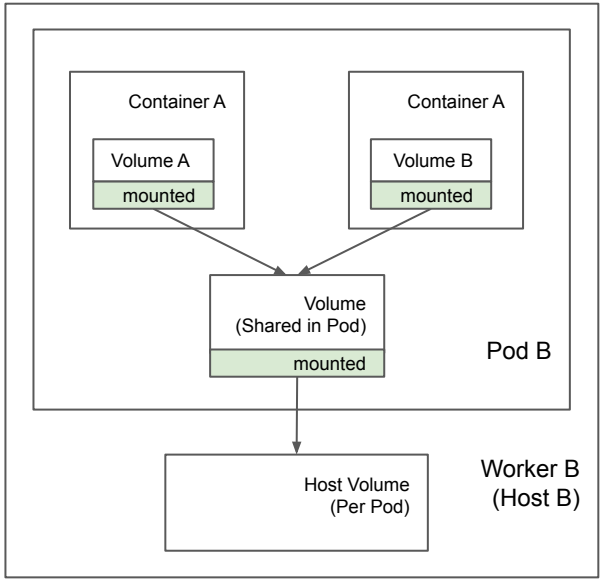

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-app-pvc
  namespace: my-app-sandbox
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

볼륨 (2/4) - Persistent Volume (ReadWriteOnce)

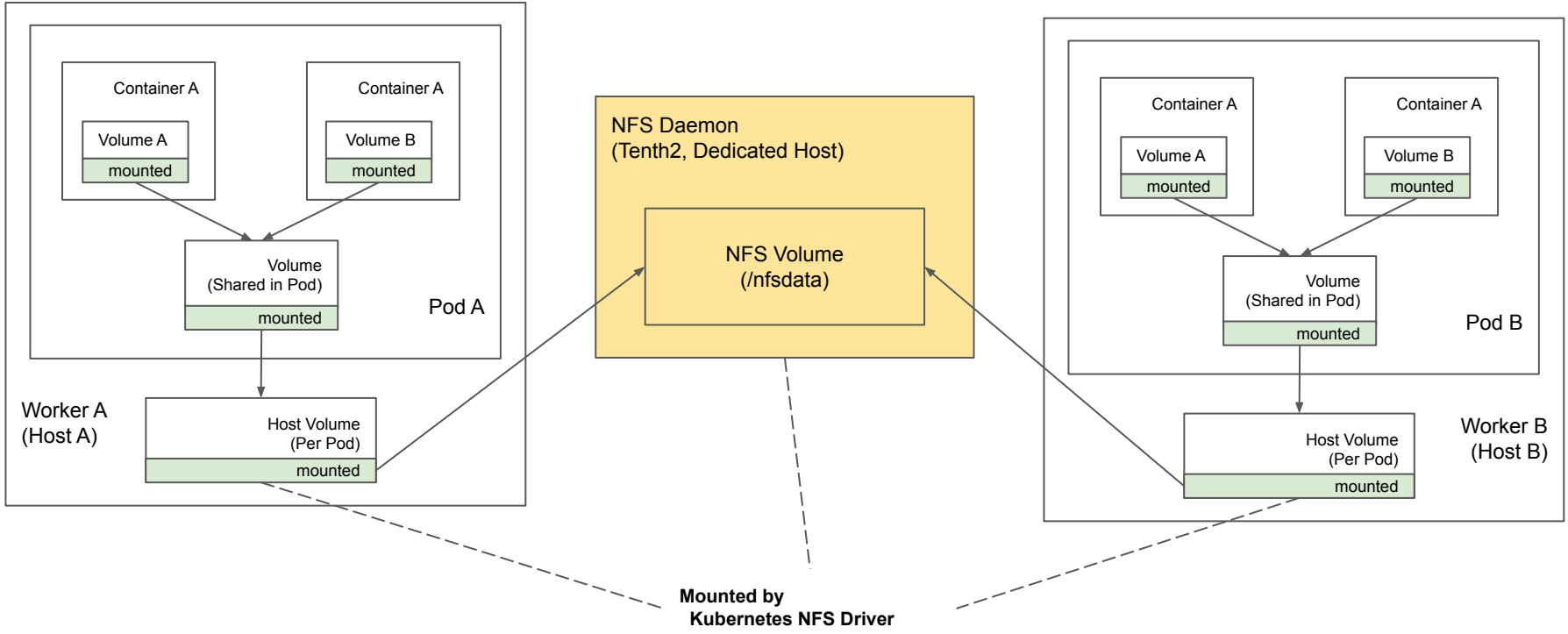


볼륨 (3/4) - Persistent Volume (ReadWriteMany)

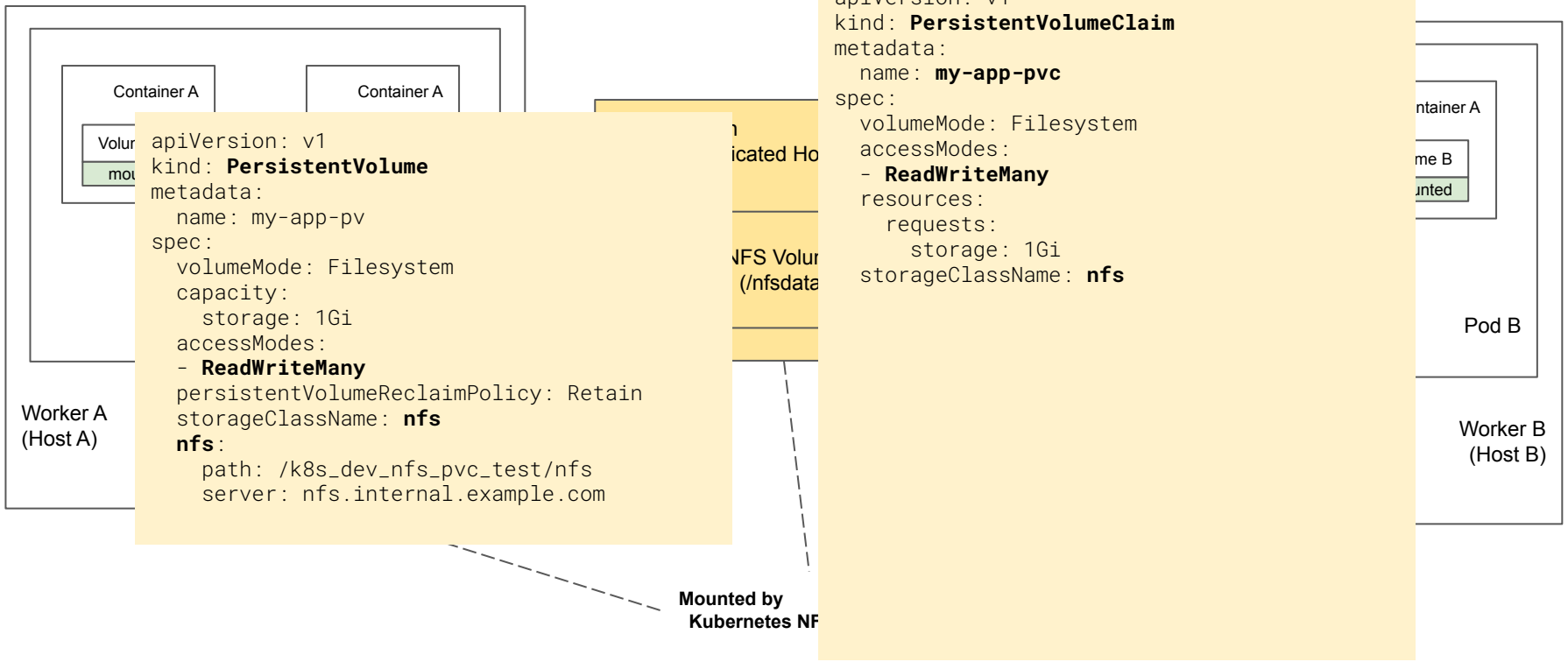




볼륨 (3/4) - Persistent Volume (ReadWriteMany)



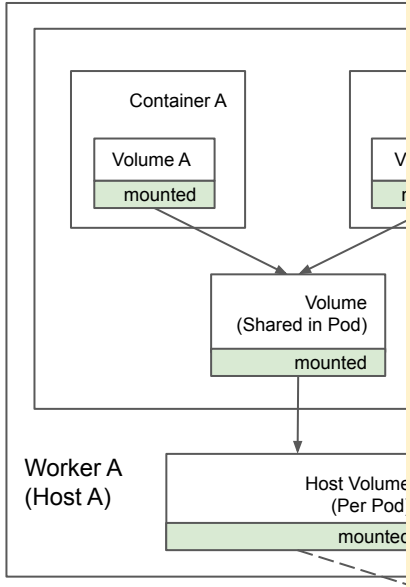
볼륨 (3/4) - Persistent Volume (ReadWriteMany)



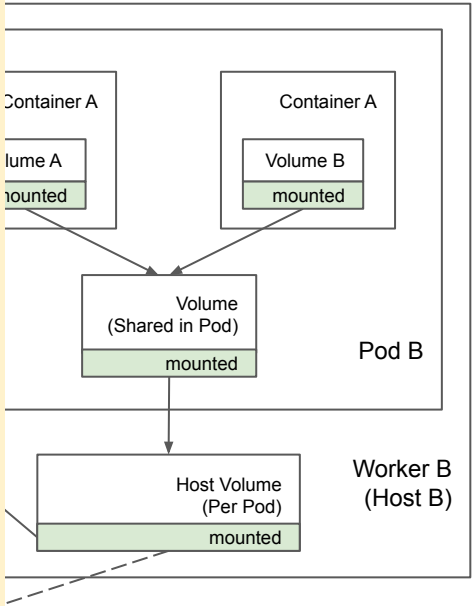

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-app-pv
spec:
  volumeMode: Filesystem
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs
  nfs:
    path: /k8s_dev_nfs_pvc_test/nfs
    server: nfs.internal.example.com
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-app-pvc
spec:
  volumeMode: Filesystem
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs
```

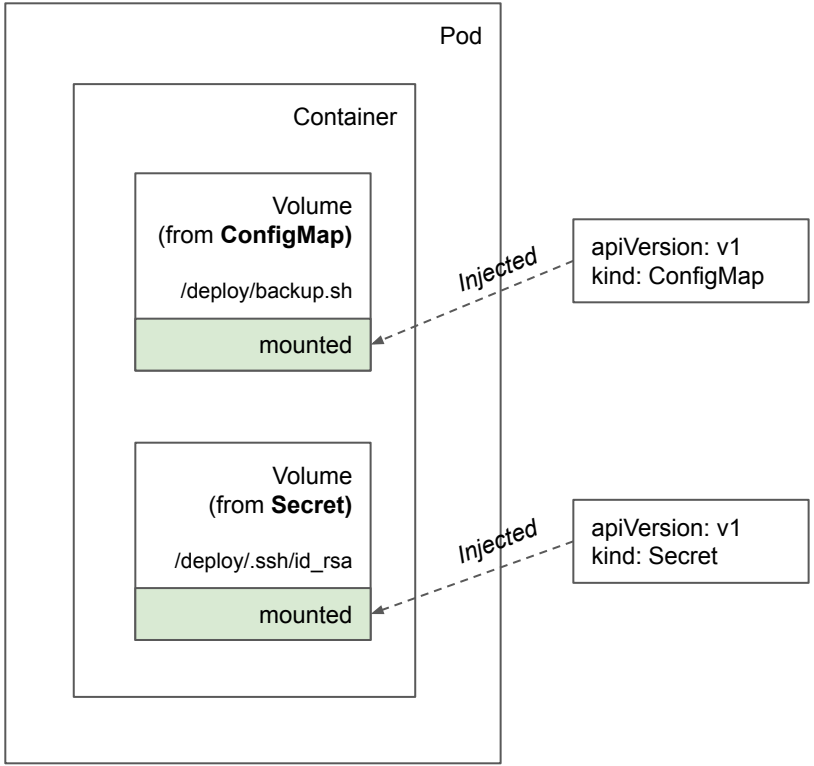
볼륨 (3/4) - Persistent Volume (ReadWriteMany)



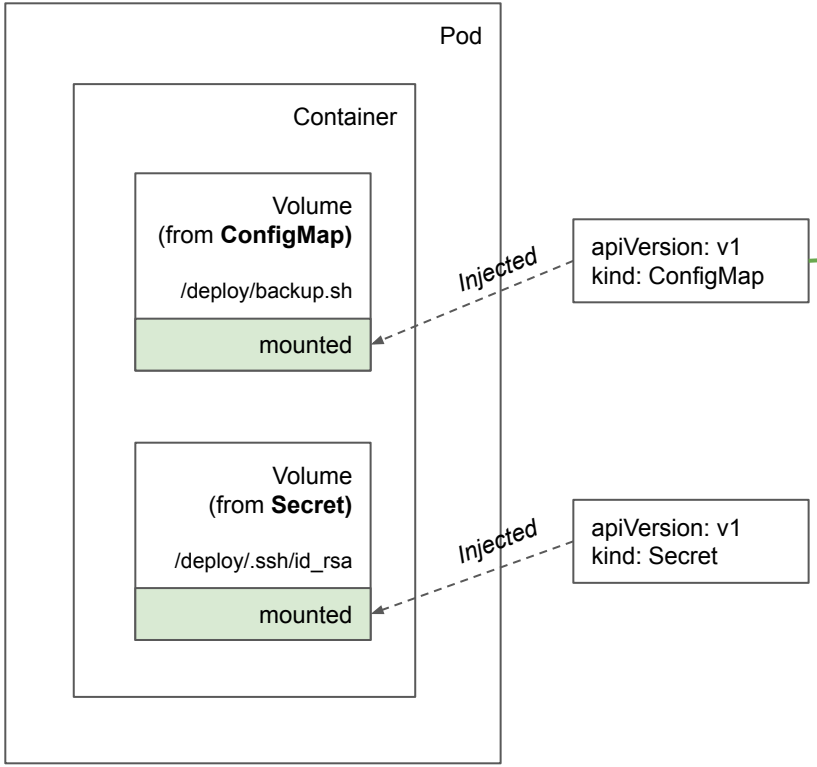

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  ...
spec:
  ...
  template:
    ...
    spec:
      containers:
        image: sangwonl/my-app:1.58.1
        name: my-app
        ...
        volumeMounts:
        - name: my-app-volume
          mountPath: "/var/lib/my-shared-data"
      volumes:
      - name: my-app-volume
        persistentVolumeClaim:
          claimName: my-app-pvc

      ...
```



볼륨 (4/4) - Configmap / Secret as Volume

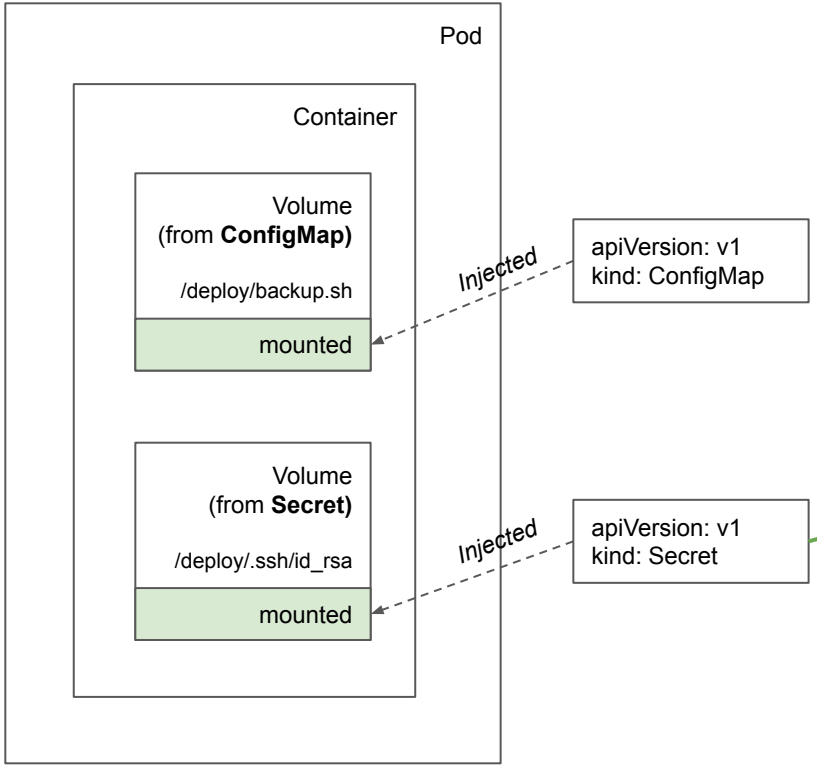


볼륨 (4/4) - Configmap / Secret as Volume



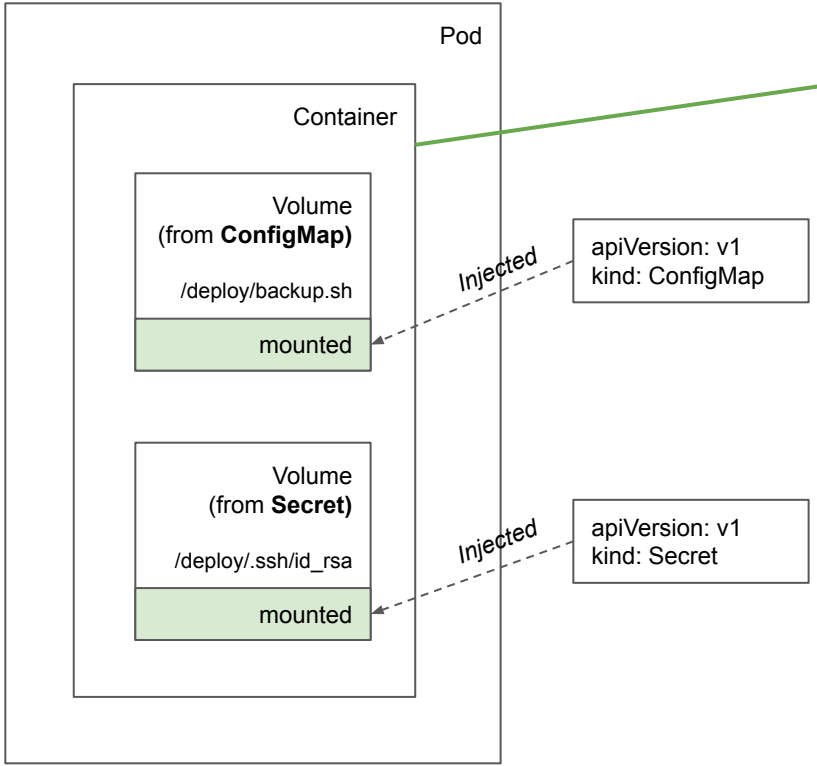

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-scripts
  namespace: my-app-sandbox
data:
  backup.sh: |
    #! /usr/bin/env bash
    mongodump \
      --host my-app-rs/my-app-mongo.example.com \
      -d my-app -o backups/

  restore.sh: |
    #! /usr/bin/env bash
    mongorestore \
      ..
```

볼륨 (4/4) - Configmap / Secret as Volume




```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-app-sshkey
  namespace: my-app-sandbox
data:
  id_rsa: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVk...
          # base64 encoded string
```

볼륨 (4/4) - Configmap / Secret as Volume



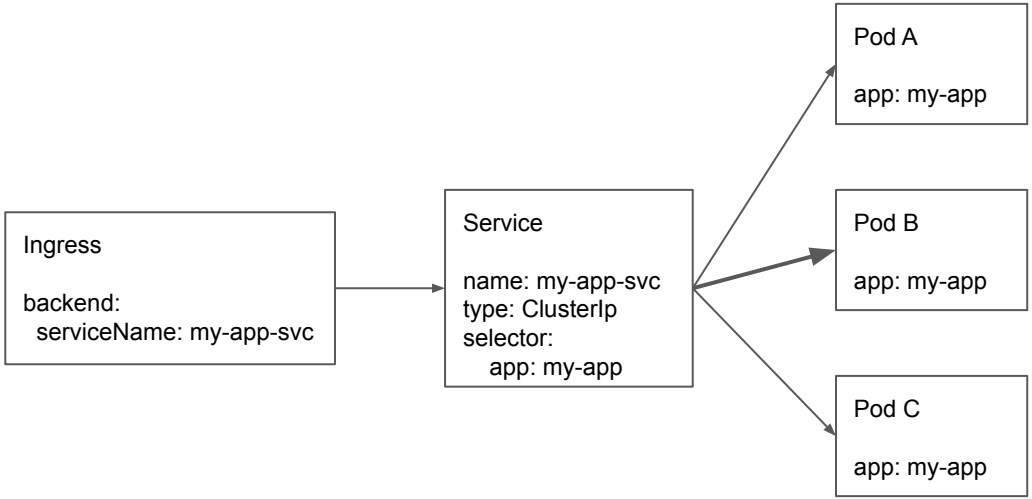

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    ...
    spec:
      containers:
        ...
        volumeMounts:
        - name: my-app-scripts-volume
          mountPath: /deploy/backup.sh
          subPath: backup.sh
        - name: my-app-sshkey-volume
          mountPath: "/deploy/.ssh"
          readOnly: true
      volumes:
        - name: my-app-scripts-volume
          configMap:
            name: my-app-scripts
            defaultMode: 0777
        - name: my-app-sshkey-volume
          secret:
            secretName: my-app-sshkey
      ...
```

크론잡 (스케쥴잡)

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: my-app-cron
spec:
  concurrencyPolicy: Forbid
  startingDeadlineSeconds: 600
  schedule: "30 2 * * *"
  jobTemplate:
    spec:
      backoffLimit: 0
      template:
        spec:
          restartPolicy: Never
          containers:
          - name: my-app-backup
            image: mongo:3.6
            command:
            - /deploy/backup.sh
            volumeMounts:
            - name: my-app-scripts-volume
              mountPath: /deploy/backup.sh
              subPath: backup.sh
          volumes:
            - name: my-app-scripts-volume
              configMap:
                name: my-app-scripts
                defaultMode: 0777
```

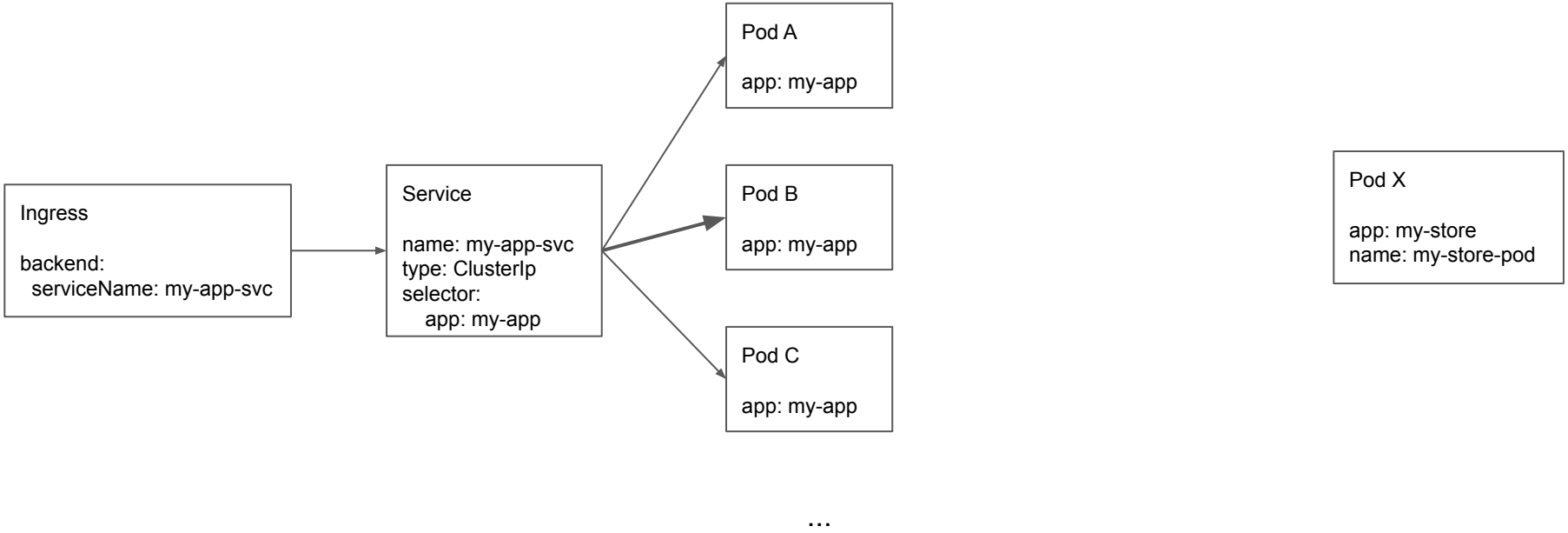
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-scripts
  namespace: my-app-sandbox
data:
  backup.sh: |
    #! /usr/bin/env bash
    mongodump \
      --host my-app-rs/my-app-mongo.example.com \
      -d my-app -o backups/

  restore.sh: |
    #! /usr/bin/env bash
    mongorestore \
      ..
```

싱글톤 Pod

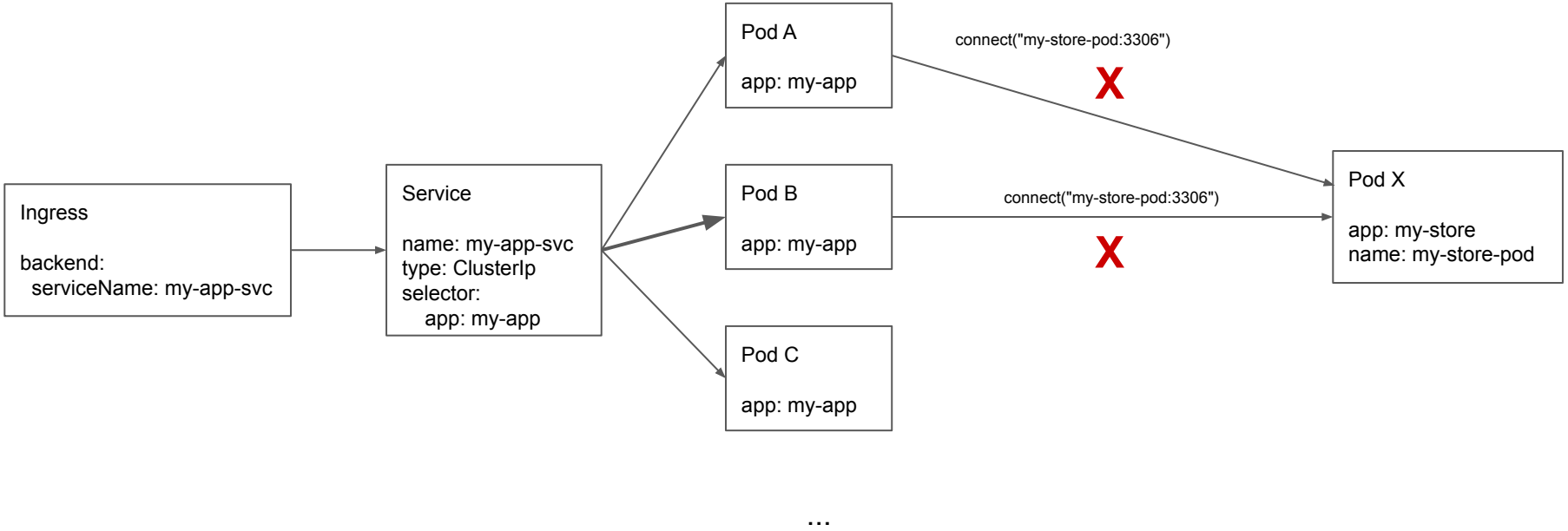

Ingress
backend:
serviceName: my-app-svc
Service
name: my-app-svc
type: ClusterIp
selector:
app: my-app
Pod A
app: my-app
Pod B
app: my-app
Pod C
app: my-app

...

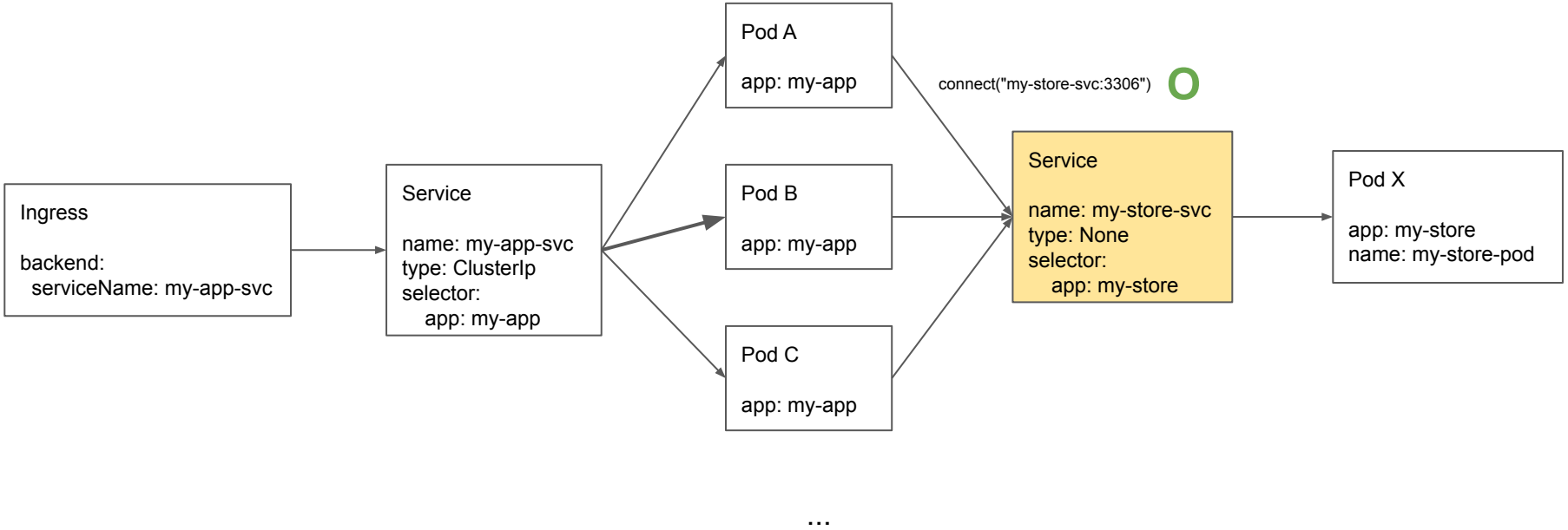
싱글톤 Pod


Ingress
backend:
serviceName: my-app-svc
Service
name: my-app-svc
type: ClusterIp
selector:
app: my-app
Pod A
app: my-app
Pod B
app: my-app
Pod C
app: my-app
...
Pod X
app: my-store
name: my-store-pod

싱글톤 Pod

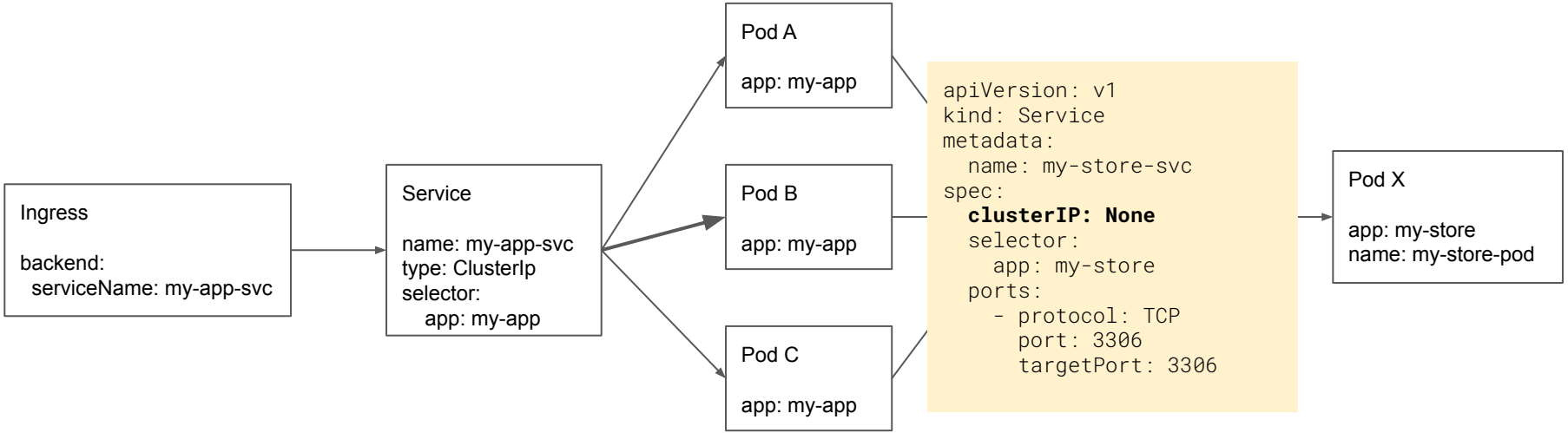

Ingress
backend:
serviceName: my-app-svc
Service
name: my-app-svc
type: ClusterIp
selector:
app: my-app
Pod A
app: my-app
Pod B
app: my-app
Pod C
app: my-app
connect("my-store-pod:3306")
X
connect("my-store-pod:3306")
X
Pod X
app: my-store
name: my-store-pod
...

싱글톤 Pod - Headless Service



...

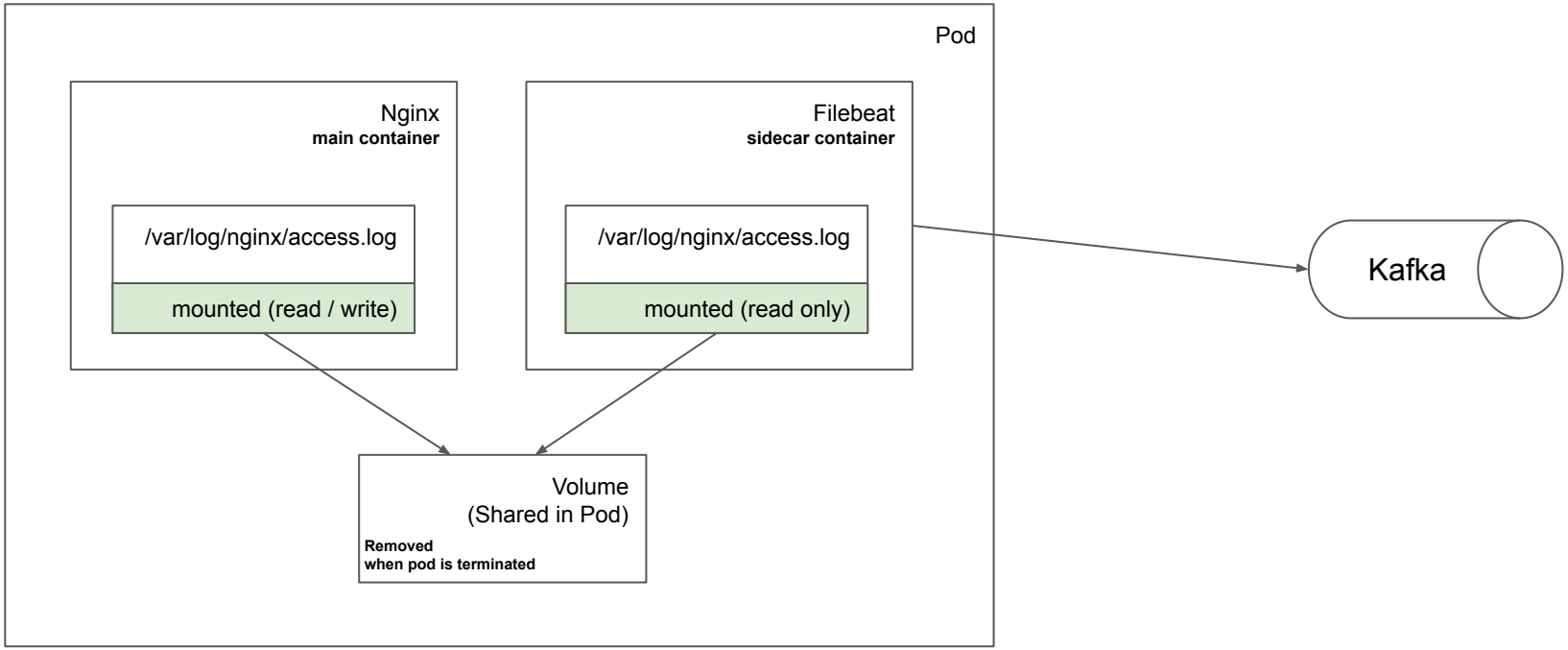
싱글톤 Pod - Headless Service



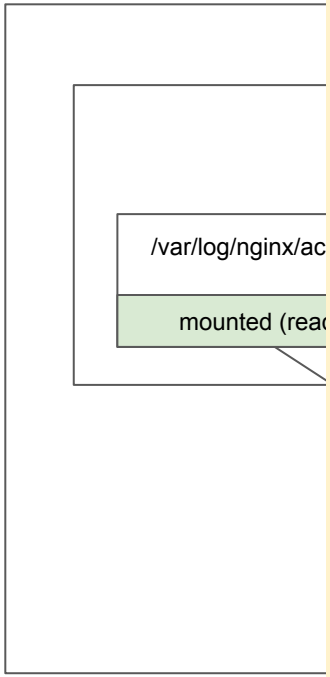
...

사이드카 (Sidecar)

사이드카 (Sidecar)



사이드카 (Sidecar)




```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: my-app
spec:
  ...
  template:
    spec:
      containers:
      - name: my-app
        image: "nginx:latest"
        ports:
        - containerPort: 80
        volumeMounts:
        - name: my-app-logs
          mountPath: /var/log/nginx
      - name: filebeat
        image: docker.elastic.co/beats/filebeat:6.2.4
        volumeMounts:
        - name: my-app-logs
          mountPath: /var/log/nginx
          readOnly: true
        command:
        - filebeat
        - -c
        - "/etc/filebeat.yml"
      volumes:
      - name: my-app-logs
        emptyDir: {}
  ...
```

4. 웹서비스 관련

HTTPS(TLS) 설정 (1/3) - HTTP

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 8000
```

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: my-app-svc
  name: my-app-svc
spec:
  ports:
  - port: 8000
    protocol: TCP
    targetPort: 8000
  selector:
    app: my-app
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-app
  name: my-app
spec:
  template:
    spec:
      containers:
      - image: my-app:latest
        name: my-app
        ports:
        - containerPort: 8000
```

HTTPS(TLS) 설정 (1/3) - TLS 설정 추가

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 8000
```

HTTPS(TLS) 설정 (1/3) - TLS 설정 추가

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 8000
  tls:
  - secretName: tls-secret
    hosts:
    - my-app.example.com
```

HTTPS(TLS) 설정 (1/3) - TLS 설정 추가

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 8000
  tls:
  - secretName: tls-secret
    hosts:
    - my-app.example.com
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.crt: LS0tLS1CRUdJTiBDRVJ...
  tls.key: LS0tLS1CRUdJTiBSU0E...
```

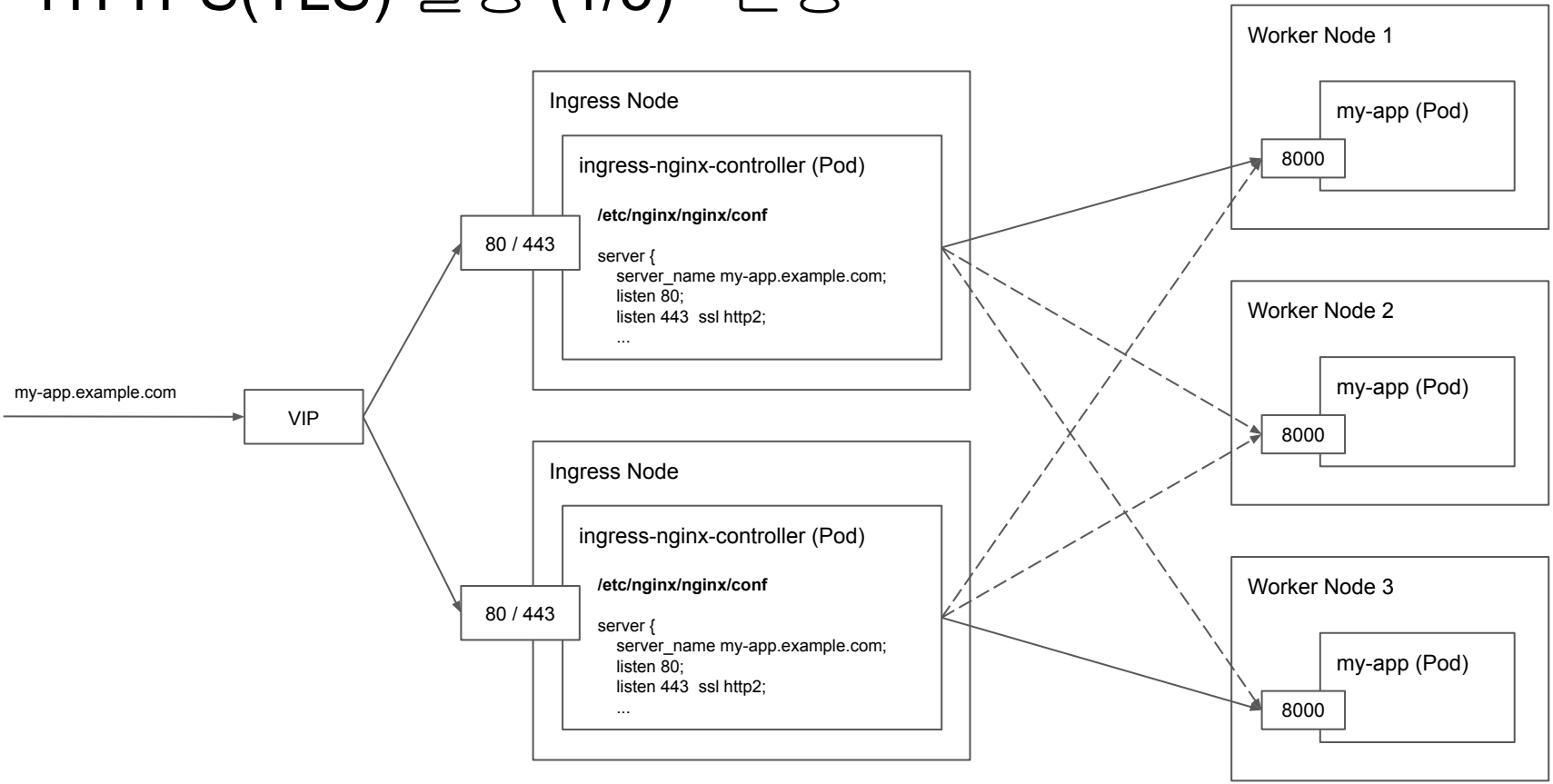
HTTPS(TLS) 설정 (1/3) - TLS 설정 추가

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 8000
  tls:
  - secretName: tls-secret
    hosts:
    - my-app.example.com
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: kubernetes.io/tls
data:
  tls.crt: LS0tLS1CRUdJTiBDRVJ...
  tls.key: LS0tLS1CRUdJTiBSU0E...
```

```
➜ $ cat STAR.example.com_crt.pem | base64
LS0tLS1CRUdJTiBDRVJ...

➜ $ cat STAR.example.com_key.pem | base64
LS0tLS1CRUdJTiBSU0E...
```

HTTPS(TLS) 설정 (1/3) - 반영



HTTPS(TLS) 설정 (2/3) - HTTP / HTTPS 동시 지원

TLS 설정을 추가하면 기본적으로는 HTTPS 만 지원함 (HTTP 로 요청하면 308 Redirect 시킴)
https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#server-side-https-enforcement-through-redirect

HTTP / HTTPS 를 모두 지원하고 싶다면 Ingress 에 Annotation을 추가

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
...
```

Ingress 는 TLS 설정 없이, 외부 LB를 통해 항상 SSL offloading 을 하고 들어오게 하려면?

```
nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
```

HTTPS(TLS) 설정 (3/3) - gRPC 지원

HTTPS(TLS) 설정 (3/3) - gRPC 지원

gRPC

HTTPS(TLS) 설정 (3/3) - gRPC 지원

gRPC $\xrightarrow{\text{over}}$ **HTTP2**

HTTPS(TLS) 설정 (3/3) - gRPC 지원

gRPC $\xrightarrow{\text{over}}$ **HTTP2** $\xrightarrow{\text{MUST use}}$ **TLS 1.2**
or higher

HTTPS(TLS) 설정 (3/3) - gRPC 지원

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "GRPC"
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 50051
  tls:
  - secretName: tls-secret
    hosts:
    - my-app.example.com
```

HTTPS(TLS) 설정 (3/3) - gRPC 지원

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redi
    nginx.ingress.kubernetes.io/backend-
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 50051
  tls:
  - secretName: tls-secret
    hosts:
    - my-app.example.com
```

```
server {
  server_name my-app.example.com;

  listen 80;
  listen 443 ssl http2;

  # Allow websocket connections
  grpc_set_header Upgrade              $http_upgrade;
  grpc_set_header Connection           $connection_upgrade;
  grpc_set_header X-Request-ID         $req_id;
  grpc_set_header X-Real-IP            $the_real_ip;
  grpc_set_header X-Forwarded-For      $the_real_ip;
  grpc_set_header X-Forwarded-Host     $best_http_host;
  grpc_set_header X-Forwarded-Port     $pass_port;
  grpc_set_header X-Forwarded-Proto    $pass_access_scheme;
  grpc_set_header X-Original-URI       $request_uri;
  grpc_set_header X-Scheme             $pass_access_scheme;
  ...
  ...

  grpc_pass grpc://upstream_balancer;

  ...
}
```

HTTPS(TLS) 설정 (3/3) - gRPC 지원

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  labels:
    app: my-app-ingress
  name: my-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redi
    nginx.ingress.kubernetes.io/backend-
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: 50051
  tls:
  - secretName: tls-secret
    hosts:
    - my-app.example.com
```

```
server {
  server_name my-app.example

  listen 80;
  listen 443 ssl http2;

  # Allow websocket connecti
  grpc_set_header Upgrade
  grpc_set_header Connection
  grpc_set_header X-Request-
  grpc_set_header X-Real-IP
  grpc_set_header X-Forwarde
  grpc_set_header X-Forwarde
  grpc_set_header X-Forwarde
  grpc_set_header X-Forwarde
  grpc_set_header X-Original
  grpc_set_header X-Scheme
  ...
  ...

  grpc_pass grpc://upstream_balancer;

  ...
}
```

nginx.org/en/docs/http/ngx_http_grpc_module.html

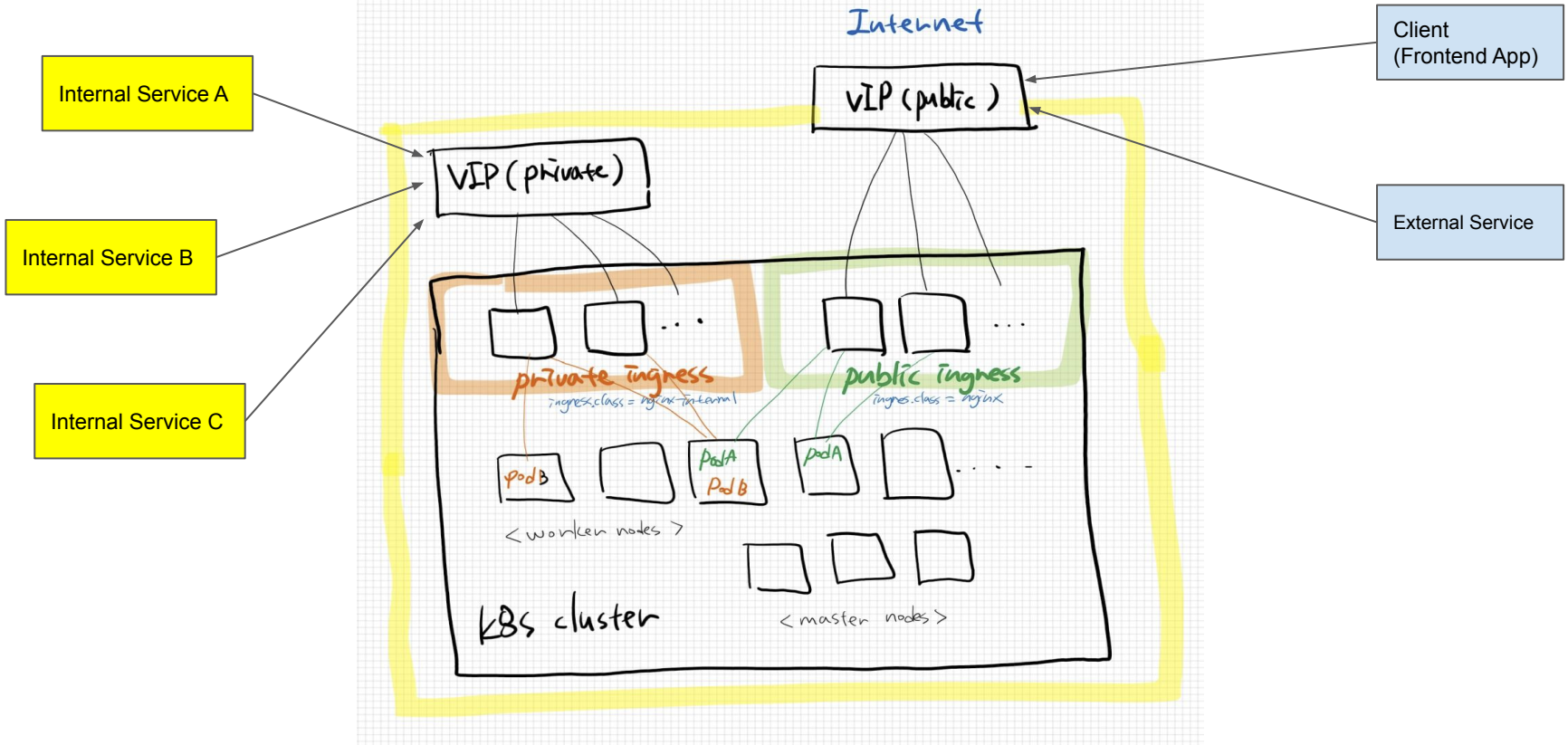
Join NGINX engineers for a livestream on NJS: Extending N
April 28 at 9:30pm IST/4:30pm CET/12:30pm E

Module ngx_http_grpc_module

Example Configuration
Directives
- grpc_bind
- grpc_buffer_size
- grpc_connect_timeout
- grpc_hide_header
- grpc_ignore_headers
- grpc_intercept_errors
- grpc_next_upstream
- grpc_next_upstream_timeout
- grpc_next_upstream_tries
- grpc_pass
- grpc_pass_header
- grpc_read_timeout
- grpc_send_timeout
- grpc_set_header

http://nginx.org/en/docs/http/ngx_http_grpc_module.html

Ingress Practice - Private / Public 용 그룹 분리 (보안)



Ingress Practice - 로깅 포맷 변경

- 가령, Ingress(Nginx) 로그를 JSON 형태로 출력하고 싶다면

```
➜ $ kubectl edit cm ingress-nginx -n ingress-nginx
```

```
apiVersion: v1
kind: ConfigMap
data:
  server-snippet: set $resp_body "";
  http-snippet: |-
    body_filter_by_lua '
      local resp_body = string.sub(ngx.arg[1], 1, 1000)
      ngx.ctx.buffered = (ngx.ctx.buffered or "") .. resp_body
      if ngx.arg[2] then
        ngx.var.resp_body = ngx.ctx.buffered
      end
    ';
  log-format-escape-json: "true"
  log-format-upstream:
'{"time":"$time_iso8601","req_id":"$req_id","remote_address":"$proxy_add_x_forwarded_for","remote_port":$remote_port,"local
_address":"$server_addr","local_port":$server_port,"service_name":"$service_name","User-Agent":"$http_user_agent","Host":"$
host","Content-Type":"$sent_http_content_type","body":"$request_body","status":$status,"body":"$resp_body","upstream_addr":
"$upstream_addr","request_time":"$request_time","upstream_response_time":"$upstream_response_time","request_method":"$reque
st_method","url":"$uri","server_protocol":"$server_protocol"}'
```

더 자세한 파라미터 참고: [Log format - NGINX Ingress Controller (kubernetes.github.io)]

Ingress Practice - 공통 설정 변경

- Ingress(Nginx) 공통 설정은 ConfigMap으로 변경

https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/

```
apiVersion: v1
kind: ConfigMap
data:
  generate-request-id: "false"
  use-forwarded-headers: "true"
  use-http2: "true"
  ...
```

컨테이너 이미지 취약점 점검 (1/3)

- 컨테이너 이미지 보안 취약점 해결 (불필요 명령어 삭제 & Package Update)

```
# 패키지 업데이트
RUN apk -y update
  # for other linux distributions
  # or apt-get -y update
  # or yum -y update


# 불필요한 패키지 제거
RUN rm \
    /usr/bin/wget \
    /usr/bin/curl \
    /usr/bin/nc \
    /sbin/route \
    /bin/ping \
    /bin/ping6
```

컨테이너 이미지 취약점 점검 (2/3)

- 컨테이너 실행 유저 non-root 로 변경

```
# Docker 를 쓰는 경우면 Dockerfile에 새로운 유저를 추가하고
# 실행 파일이 있거나 실행 중 필요한 경로에 대해 권한을 주는 명령을 추가

RUN addgroup deploy && adduser -D -G deploy deploy \
    && chown -R deploy:deploy $DEPLOY_HOME \
    && chmod -R 740 $DEPLOY_HOME

USER deploy
```

컨테이너 이미지 취약점 점검 (3/3)

- 컨테이너 실행 유저 non-root 로 변경 (JIB)

```
# Gradle 설정을 통해 jib 을 쓰는 경우라면
# 이전장에서처럼 nonroot 유저가 추가된 이미지를 하나 생성한 후에
# jib.container.user 로 설정

jib {
    from {
        image = 'sangwonl/java-debian10:nonroot'
    }
    container {
        user = 'deploy'
        environment = []
        jvmFlags = []
    }
}
```

라이브 중인 기존 서비스 k8s 클러스터로 이전

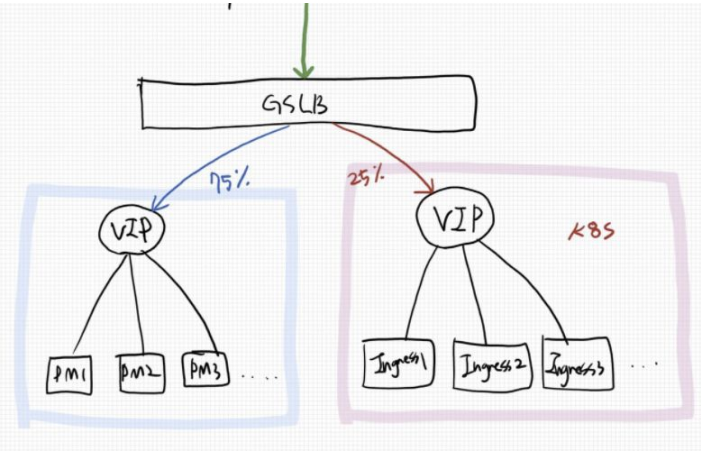
- 작업 중 가장 부담스러운 부분, 점진적으로 트래픽을 이전하는 방법이 없을까?

라이브 중인 기존 서비스 k8s 클러스터로 이전

- 작업 중 가장 부담스러운 부분, 점진적으로 트래픽을 이전하는 방법이 없을까?

- LB와 도메인 사이에 중간 LB를 둬서 일정 비율로 라우팅을 전환하는 방법을 권장
  (Provision 가능한 LB 면 좋지만 그게 아니면 Nginx 등으로 직접 구성도 가능)

- 신규 클러스터의 트래픽 유입 비율을 점진적으로 전환 가능
  (가령, 1% → 5% → 25% → 50% → 75% → 100%)



라이브 중인 기존 서비스 k8s 클러스터로 이전 - 유의사항

- 혹시 새로운 LB-IP로 도메인을 이전했다면 중간 단계 DNS 혹은 클라이언트 DNS 캐싱에서 캐싱 가능성

- 따라서 기존 LB-IP가 도메인에서 완전히 제거되었는지, 트래픽 유입이 없는지 확인하시고 삭제하길 권장

- Ingress의 Routing Rule과 TLS 설정(시크릿) 등을 한번 더 확인하세요! (실수 많이 일어나는 부분)

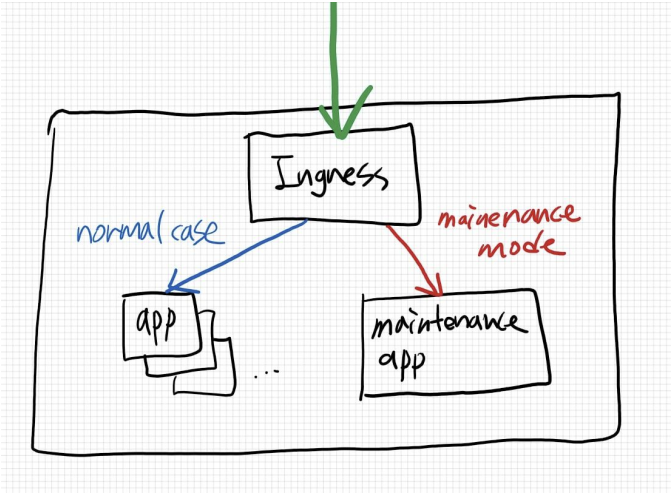
```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"      # 분리한 외부/내부 Ingress 클래스가 맞는지!
  name: my-public-service-ingress
spec:
  rules:
  - host: my-service.example.com              # 적용하려는 도메인이 맞는지!
    http:
      paths:
      - path: /
        backend:
          serviceName: my-public-service
          servicePort: http-port
  tls:
  - secretName: tls-example-com               # 해당 도메인의 인증서가 맞는지!
    hosts:
    - my-service.example.com
```

서비스(프론트) 점검 걸기 (1/3)

- 배포시에 유저 유입 막기 위한 점검 페이지 띄우기

- Proxy(Nginx)에서 점검페이지를 띄우고 특정 Source IP는 Allow 해주는 방식

- k8s Ingress도 결국은 Nginx와 같은 Proxy 서버이기 때문에 Ingress 설정으로 가능



서비스(프론트) 점검 걸기 (2/3) - 점검 서비스를 별도로

Allow 하지 않은 source에서 오는 요청들을 403 → 이때 점검 서비스로 proxy_pass

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/whitelist-source-range: 172.0.0.0/8,127.0.0.1
    nginx.ingress.kubernetes.io/server-snippet: |
      error_page 403 @maintenance;                    # 위의 whitelist-source-range를 통해 제한된 유저들은 403 응답이 내려가는데 이를
캐치해서
      location @maintenance {
        proxy_pass http://my-app-repair.example.com;  # maintenance 서비스로 proxy_pass 한 결과로 응답
      }
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: http-port
```

server-snippet 참고

서비스(프론트) 점검 걸기 (3/3) - 프론트앱에서 점검도 제공

Allow 하지 않은 source에서 오는 요청들을 403 → 이때 원래 서비스로 proxy_pass
(단, 프론트앱에서 점검 상태인지를 알 수 있도록 헤더 제공)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/whitelist-source-range: 172.0.0.0/8,127.0.0.1/32
    nginx.ingress.kubernetes.io/server-snippet: |
      error_page 403 @maintenance;                # 위의 whitelist-source-range를 통해 제한된 유저들은 403 응답이 내려가는데 이를 캐치해서
      location @maintenance {
        proxy_pass http://upstream_balancer;      # 이 Ingress의 원래 서비스(my-app-svc)로 그대로 넘기는데
        proxy_set_header X-Maintenance "on";      # X-Maintenance: "on" 이라는 헤더를 추가해서 넘김 (그럼 앱에서 이 헤더를 보고 분기 가능)
      }
  name: my-app-ingress
spec:
  rules:
  - host: my-app.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: my-app-svc
          servicePort: http-port
```

서비스(프론트) 점검 걸기 (3/3) - 프론트앱에서 점검도 제공

프론트 앱에서 점검 플래그 헤더를 확인해서 점검 페이지를 노출

Nginx 예시

```
if ($http_x_maintenance = "on") {
    return 501;
}

error_page 501 @maintenance;

location @maintenance {
    root /etc/nginx/html;
    rewrite ^(.*)$ /maintenance.html break;
}
```

5. 배포 파이프라인

ArgoCD 활용

- GitOps (https://www.gitops.tech/#what-is-gitops)

The core idea of GitOps is having a Git repository that always contains **declarative descriptions** of the infrastructure currently desired in the production environment and an automated process to make the production environment match the described state in the repository.

ArgoCD 활용

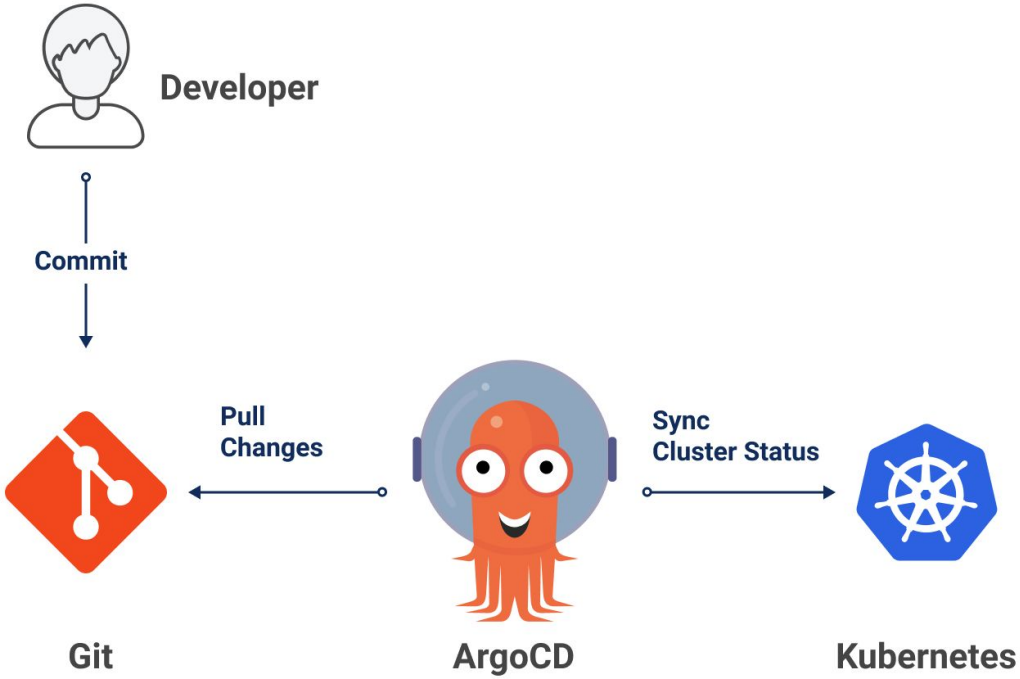
- GitOps (https://www.gitops.tech/#what-is-gitops)

The core idea of GitOps is having a Git repository that always contains **declarative descriptions** of the infrastructure currently desired in the production environment and an automated process to make the production environment match the described state in the repository.

- Tools: FluxCD vs ArgoCD vs JenkinsX

Capability	Flux	ArgoCD	Jenkins X
Sync Git to cluster	:)	:)	:)
Push app changes to Git	:|		:)
Handle full CI/CD			:)
Multi team support		:)	
Self-service adding of new repo			
Helm support	:)	:)	:)
Kustomize support	:)	:)	

ArgoCD 활용


Developer
Commit
Pull
Changes
Sync
Cluster Status
Git
ArgoCD
Kubernetes

ArgoCD 활용 - 설치

- ArgoCD 설치를 참고 (핵심은 install.yaml 파일이고 이걸 기준으로 커스터마이징 및 확장)

```
$ kubectl create namespace argocd

$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

- CLI 설치

```
$ brew install argocd
```

ArgoCD 활용 - 커스터마이징

- LDAP 설정을 위한 ConfigMap

```yaml
# argocd-cm.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
data:
  statusbadge.enabled: 'true'
  url: https://argocd.example.com
  accounts.jenkins: apiKey
  accounts.jenkins.enabled: 'true'
  dex.config: |
    connectors:
    - type: ldap
      id: ldap
      name: LDAP
      config:
        host: ldap.example.com:389
        insecureNoSSL: true
        startTLS: false
        bindDN: ...
        bindPW: ...
        userSearch:
          baseDN: 'o=identitymaster'
          username: uid
          idAttr: uid
          emailAttr: mail
          nameAttr: uid
```

ArgoCD 활용 - 커스터마이징

- RBAC 설정을 위한 ConfigMap

```yaml
# argocd-rbac-cm.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-rbac-cm
data:
  policy.default: role:admin
```

- HTTPS & GRPC를 위해 Secret 추가

```yaml
# secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret-example-com
type: kubernetes.io/tls
data:
  tls.crt: LS0tLS1CRUdJTiBDRVJUS***
  tls.key: LS0tLS1CRUdJTiBSU0EgU***
```

ArgoCD 활용 - 커스터마이징

- Ingress 설정

```yaml
# ingress-https.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: argocd-server-http-ingress
  namespace: argocd
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: argocd-server
          servicePort: http
    host: argocd.example.com
  tls:
  - secretName: tls-secret-example-com
    hosts:
    - argocd.example.com
```

ArgoCD 활용 - 커스터마이징

- gRPC 설정

```yaml
# ingress-grpc.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: argocd-server-grpc-ingress
  namespace: argocd
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/backend-protocol: "GRPC"
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: argocd-server
          servicePort: https
    host: argocd-grpc.example.com
  tls:
  - hosts:
    - argocd-grpc.example.com
    secretName: argocd-secret
```

ArgoCD 활용 - CLI 사용

- localhost → k8s argocd server 포워딩 설정

```
$ kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

ArgoCD 활용 - CLI 사용

- k8s 클러스터 조회 / 추가 / 삭제

```
# 조회
$ argocd cluster list
SERVER                                                      NAME             VERSION  STATUS      MESSAGE
https://my-dev-04-master-1.example.com:6443                 my-dev-04-ctx    1.15     Successful
https://my-prod-03-master-1.example.com:6443                my-prod-03-ctx   1.15     Successful

# 추가
$ argocd cluster add my-devops-ctx
INFO[0000] ServiceAccount "argocd-manager" already exists in namespace "kube-system"
INFO[0000] ClusterRole "argocd-manager-role" updated
INFO[0000] ClusterRoleBinding "argocd-manager-role-binding" updated
Cluster 'https://my-devops-master-1.example.com:6443' added

# 다시 조회
$ argocd cluster list
SERVER                                                      NAME             VERSION  STATUS      MESSAGE
https://my-dev-04-master-1.example.com:6443                 my-dev-04-ctx    1.15     Successful
https://my-prod-03-master-1.example.com:6443                my-prod-03-ctx   1.15     Successful
https://my-devops-master-1.example.com:6443                 my-devops-ctx    1.11     Successful

# 삭제 - 서버이름으로 해야함
$ argocd cluster rm https://my-devops-master-1.example.com:6443

# 다시 조회
$ argocd cluster list
SERVER                                                      NAME             VERSION  STATUS      MESSAGE
https://my-dev-04-master-1.example.com:6443                 my-dev-04-ctx    1.15     Successful
https://my-prod-03-master-1.example.com:6443                my-prod-03-ctx   1.15     Successful
```

ArgoCD 활용 - CLI 사용

- Git 리파지토리 접근 Credential 관리

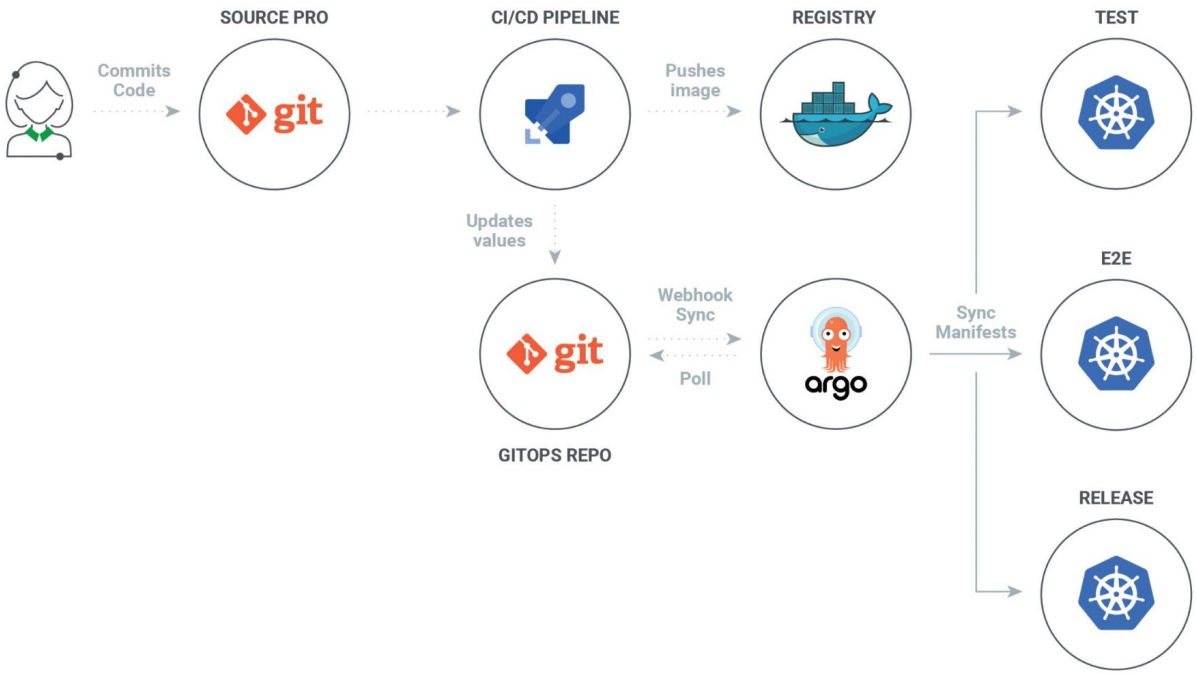
```
# 조회
$ argocd repocreds list
URL PATTERN                       USERNAME  SSH_CREDS  TLS_CREDS
https://github.com/sangwonl       sangwonl  false      false

# 추가
$ argocd repocreds add https://github.com/sangwonl --username sangwonl --password "*********"

# 삭제
$ argocd repocreds rm https://github.com/sangwonl
```

ArgoCD 활용 - CLI 사용

- Git 리파지토리 조회 / 추가 / 삭제

```
# 조회
$ argocd repo list
TYPE  NAME             REPO                                           INSECURE  OCI    LFS    CREDS  STATUS      MESSAGE
git                    https://github.com/sangwonl/kube-recipes.git   false     false  false  true   Successful

# 추가
$ argocd repo add https://github.com/sangwonl/k8s-apps.git

# 다시 조회
$ argocd repo list
TYPE  NAME             REPO                                           INSECURE  OCI    LFS    CREDS  STATUS      MESSAGE
git                    https://github.com/sangwonl/kube-recipes.git   false     false  false  true   Successful
git                    https://github.com/sangwonl/k8s-apps.git       false     false  false  true   Successful

# 삭제
$ argocd repo rm https://github.com/sangwonl/k8s-apps.git
```

ArgoCD 활용 - Pipeline



끝.

감사합니다.

Appendix

Pets vs Cattle

Pod?

Figure 8. ALQ-99 Tactical Jamming Pods