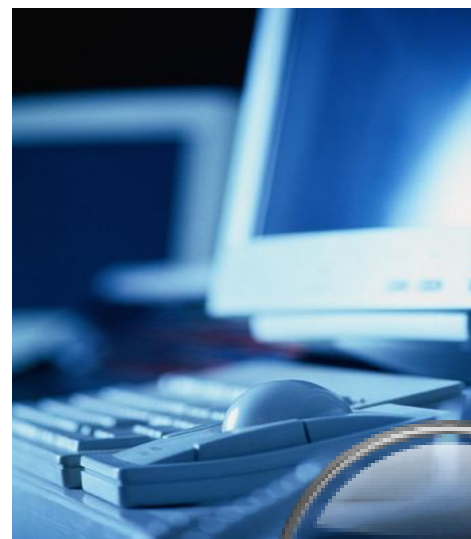


쿠버네티스 핵심 개념

- ▶ 큐브 시스템 컴포넌트
- ▶ ETCD의 개념과 사용
- ▶ 포드
- ▶ 레이블과 셀렉터
- ▶ 레플리카셋
- ▶ 디플로이먼트
- ▶ 네임스페이스
- ▶ 서비스(+인그레스)
- ▶ 네트워크 시스템
- ▶ 스토리지(hostpath, PV, PVC)

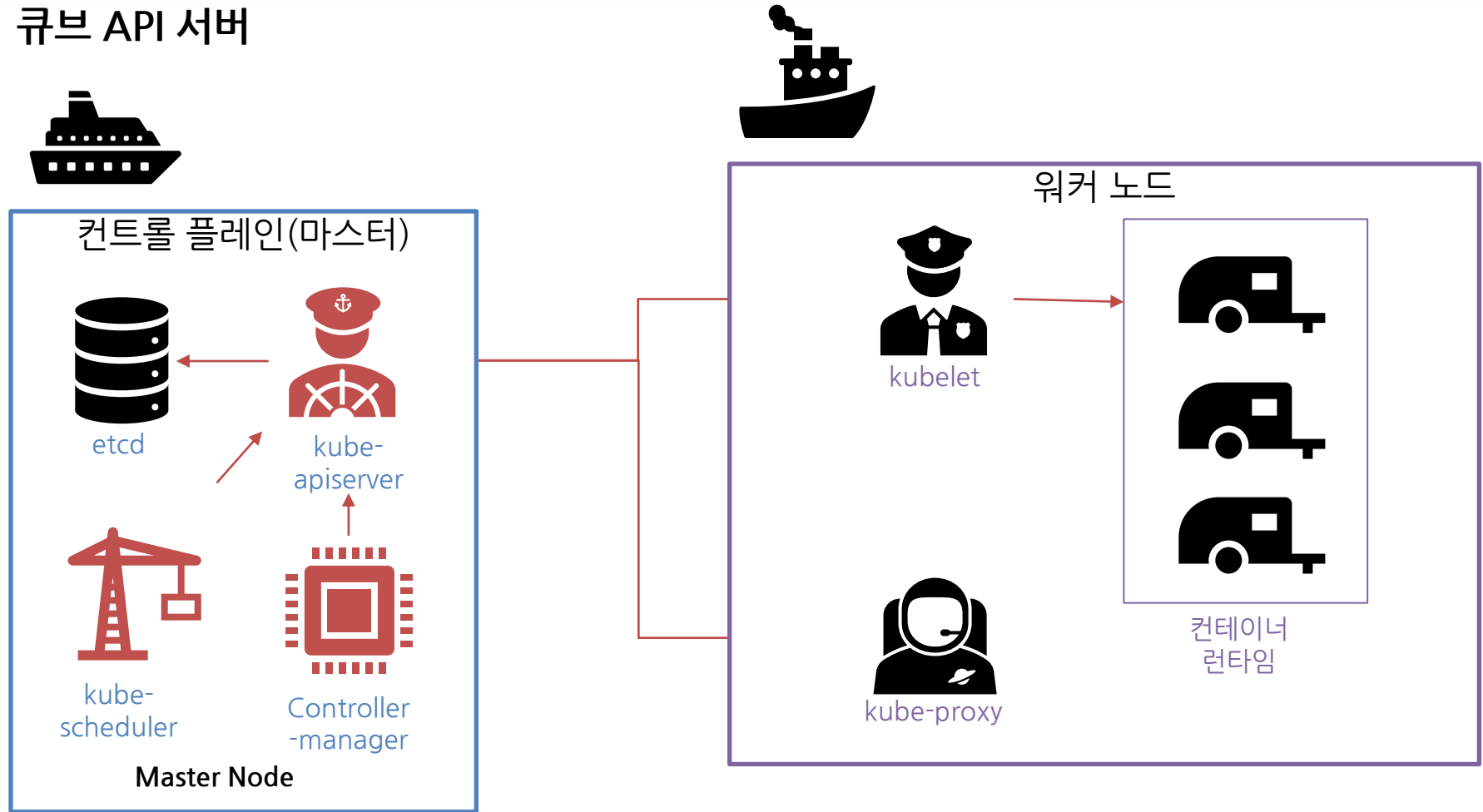




Kube-system Component

큐브 시스템 컴포넌트

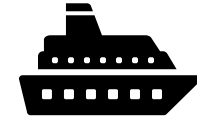
큐브 API 서버



큐브 시스템 컴포넌트

큐브 API 서버

- 쿠버네티스 시스템 컴포넌트는 오직 API 서버와 통신
- 컴포넌트끼리 서로 직접 통신 X
- 때문에 etcd와 통신하는 유일한 컴포넌트 API 서버
- RESTful API를 통해 클러스터 상태를 쿼리, 수정할 수 있는 기능 제공
- API 서버의 구체적인 역할
 - 인증 플러그인을 사용한 클라이언트 인증
 - 권한 승인 플러그인을 통한 클라이언트 인증
 - 승인 제어 플러그인을 통해 요청 받은 리소스를 확인/수정
 - 리소스 검증 및 영구 저장



컨트롤 플레인(마스터)



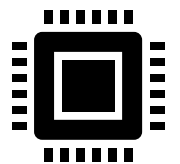
etcd



kube-
apiserver



kube-
scheduler



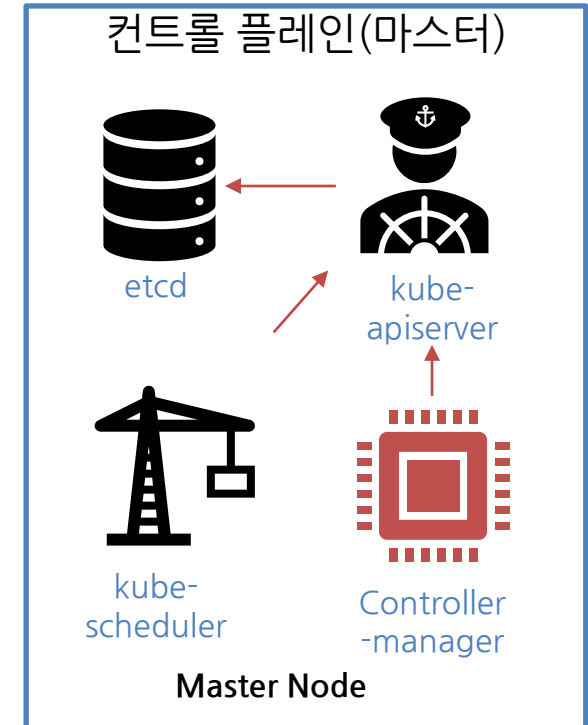
Controller-
manager

Master Node

큐브 시스템 컴포넌트

큐브 컨트롤러 매니저

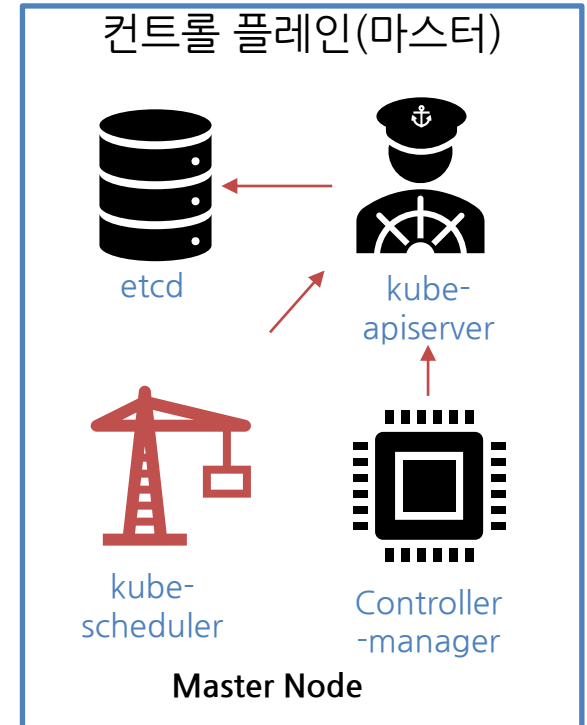
- API 궁극적으로 아무 역할을 하지 않음
- 컨트롤러에는 다양한 컨트롤러가 존재
- 이 컨트롤러는 API에 의해 받아진 요청을 처리하는 역할
 - 레플리케이션 매니저(레플리케이션컨트롤러)
 - 레플리카셋, 데몬셋, 잡 컨트롤러
 - 디플로이먼트 컨트롤러
 - 스테이트풀셋 컨트롤러,
 - 노드 컨트롤러
 - 서비스 컨트롤러
 - 엔드포인트 컨트롤러
 - 네임스페이스 컨트롤러
 - 영구 볼륨 컨트롤러
 - etc



큐브 시스템 컴포넌트

큐브 스케줄러

- 일반적으로 실행할 노드를 직접 정해주지 않음
- 요청 받은 리소스를 어느 노드에 실행할지 결정하는 역할
- 현재 노드의 상태를 점검하고 최상의 노드를 찾아 배치
- 다수의 포드를 배치하는 경우에는 라운드로빈을 사용하여 분산



Kube-system Component

쿠버네티스 주요 컴포넌트 확인하기

- `kubectl get pod -n kube-system`

```
$ kubectl get pod -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-6955765f44-b7bm8	0/1	ContainerCreating	0	8m44s
coredns-6955765f44-vtq2p	0/1	ContainerCreating	0	8m44s
etcd-master	1/1	Running	0	8m47s
kube-apiserver-master	1/1	Running	0	8m47s
kube-controller-manager-master	1/1	Running	0	8m47s
kube-proxy-9whjc	1/1	Running	0	2m45s
kube-proxy-d8cnw	1/1	Running	0	8m44s
kube-proxy-lnvwm	1/1	Running	0	8m13s
kube-scheduler-master	1/1	Running	0	8m47s

Kube-system Component

쿠버네티스 설정 파일 확인하기

- 일반적으로 리눅스에 설치하면 /etc/kubernetes/manifest에 설정 파일 존재
- GCP의 경우에는 /home/Kubernetes/kube-manifests/Kubernetes/gci-trusty에 설정 파일 존재

```
$ ls
```

abac-authz-policy.jsonl	gke-internal-configure-helper.sh	kube-proxy
addon-manager	glbc.manifest	kube-proxy.manifest
calico-policy-controller	health-monitor.sh	kube-scheduler.manifest
cluster-autoscaler.manifest	internal-capacity-request-crd.yaml	limit-range
cluster-loadbalancing	internal-cluster-autoscaler.manifest	loadbalancing
cluster-monitoring	internal-vpa-admission-controller.manifest	metadata-agent
dashboard	internal-vpa-crd.yaml	metadata-proxy
device-plugins	internal-vpa-rbac.yaml	metrics-server
dns	internal-vpa-recommender.manifest	node-problem-detector
dns-horizontal-autoscaler	internal-vpa-updater.manifest	node-termination-handler
etcd-empty-dir-cleanup.yaml	ip-masq-agent	podsecuritypolicies
etcd.manifest	istio	prometheus
fluentd-elasticsearch	kube-addon-manager.yaml	rbac
fluentd-gcp	kube-apiserver.manifest	runtimeclass
gci-configure-helper.sh	kube-controller-manager.manifest	storage-class

Kube-system Component

▶ 쿠버네티스 설정 파일 확인하기

- 일반적으로 리눅스에 설치하면 /etc/kubernetes/manifest에 설정 파일 존재
- GCP의 경우에는 /home/Kubernetes/kube-manifests/Kubernetes/gci-trusty에 위치

```
$ ps -aux | grep kubelet
root      1122   2.1   2.7 760796 103404 ?        Ssl  06:27   2:46 /home/kubernetes/bin/kubelet \
--v=2 \
--cloud-provider=gce \
--experimental-check-node-capabilities-before-mount=true \
--allow-privileged=true \
--experimental-mounter-path=/home/kubernetes/containerized_mounter/mounter \
--cert-dir=/var/lib/kubelet/pki/ \
--cni-bin-dir=/home/kubernetes/bin \
--kube config=/var/lib/kubelet/kube config \
--experimental-kernel-memcg-notification=true \
--max-pods=110 \
--network-plugin=kubenet \
--node-labels=beta.kubernetes.io/fluentd-ds-ready=true,cloud.google.com/gke-nodepool=default-
pool,cloud.google.com/gke-os-distribution=cos --volume-plugin-dir=/home/kubernetes/flexvolume \
--registry-qps=10 \
--registry-burst=20 \
--bootstrap-kube config=/var/lib/kubelet/bootstrap-kube config \
--node-status-max-images=25 \
--config /home/kubernetes/kubelet-config.yaml
```

ETCD의 개념과 사용


ETCD의 개념과 사용

ETCD?



A distributed, reliable key-value store for the most critical data of a distributed system

 Docs

 GitHub

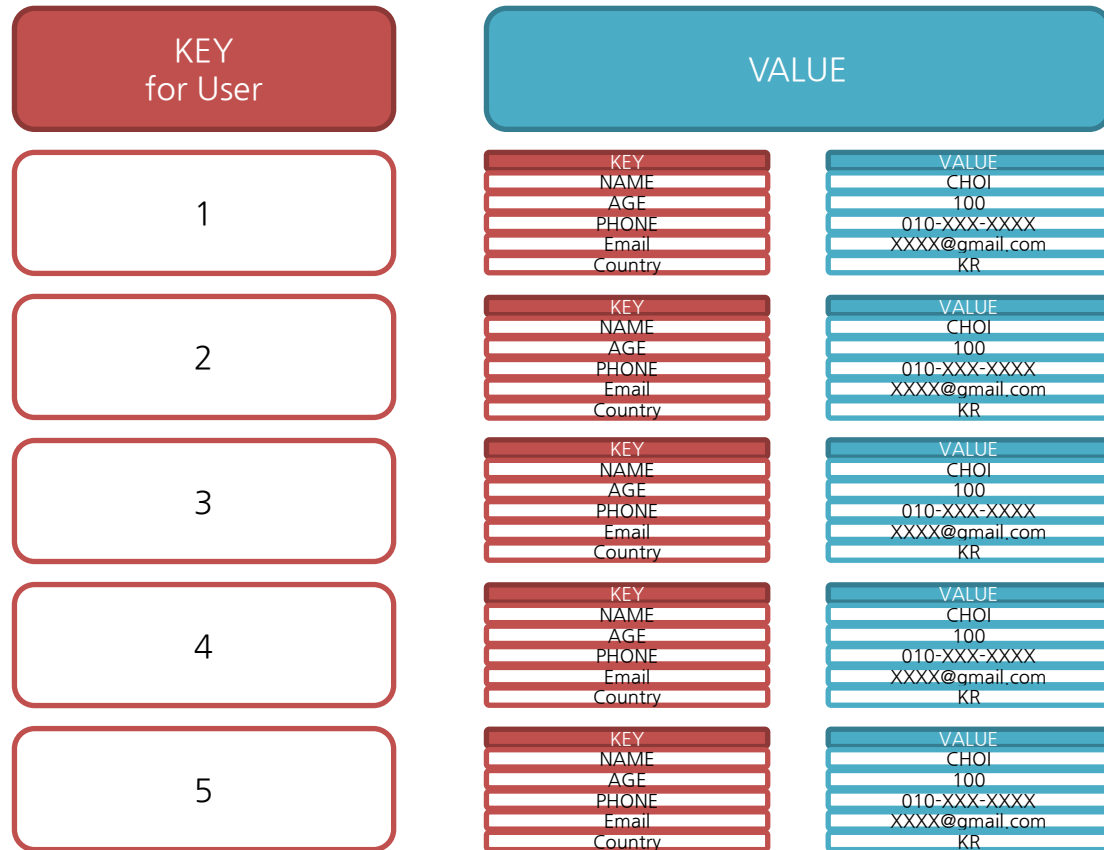
ETCD의 개념과 사용

Key-Value 데이터 셋

KEY	VALUE
NAME	CHOI
AGE	100
PHONE	010-XXX-XXXX
Email	XXXX@gmail.com
Country	KR

ETCD의 개념과 사용

다중의 Key-Value 데이터 셋



ETCD의 개념과 사용

▶ 다중의 Key-Value 데이터 셋

- <https://github.com/etcd-io/etcd/releases>

```
$ wget https://github.com/etcd-io/etcd/releases/download/v3.3.13/etcd-v3.3.13-linux-arm64.tar.gz # 파일 다운로드
```

```
$ tar -xf etcd-v3.3.13-linux-arm64.tar.gz # 압축 해제
```

```
$ cd ./etcd-v3.3.13-linux-arm64 # 파일 안에 etcdctl 명령이 존재
```

```
$ sudo ETCDCTL_API=3 ./etcdctl --endpoints 127.0.0.1:2379 --cacert  
/etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key  
/etc/kubernetes/pki/etcd/server.key get / --prefix --keys-only
```

ETCD의 개념과 사용

▶ 다중의 Key-Value 데이터 셋

- key와 value를 넣고 빼는 API 사용

```
$ sudo ETCDCTL_API=3 ./etcdctl --endpoints 127.0.0.1:2379 --cacert  
/etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key  
/etc/kubernetes/pki/etcd/server.key put key1 value1 # key value 넣기  
OK
```

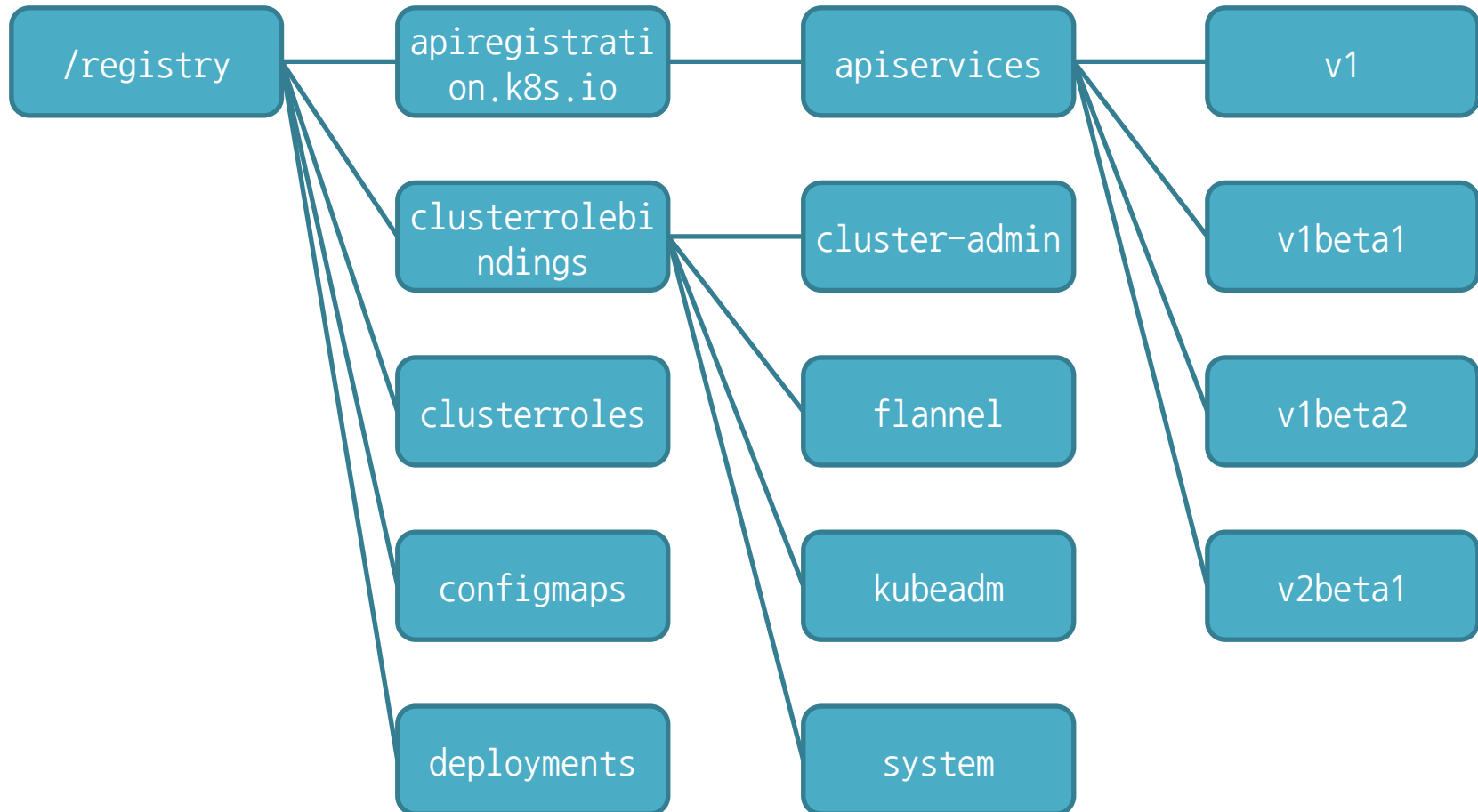
```
$ sudo ETCDCTL_API=3 ./etcdctl --endpoints 127.0.0.1:2379 --cacert  
/etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key  
/etc/kubernetes/pki/etcd/server.key get key1 # key를 사용해 value 얻기  
key1  
value1
```

ETCD의 개념과 사용

다중의 Key-Value 데이터 셋

● 쿠버네티스-ETCD 데이터베이스 키 구조

- ETCD안에 쿠버네티스의 전체 설정 정보를 저장

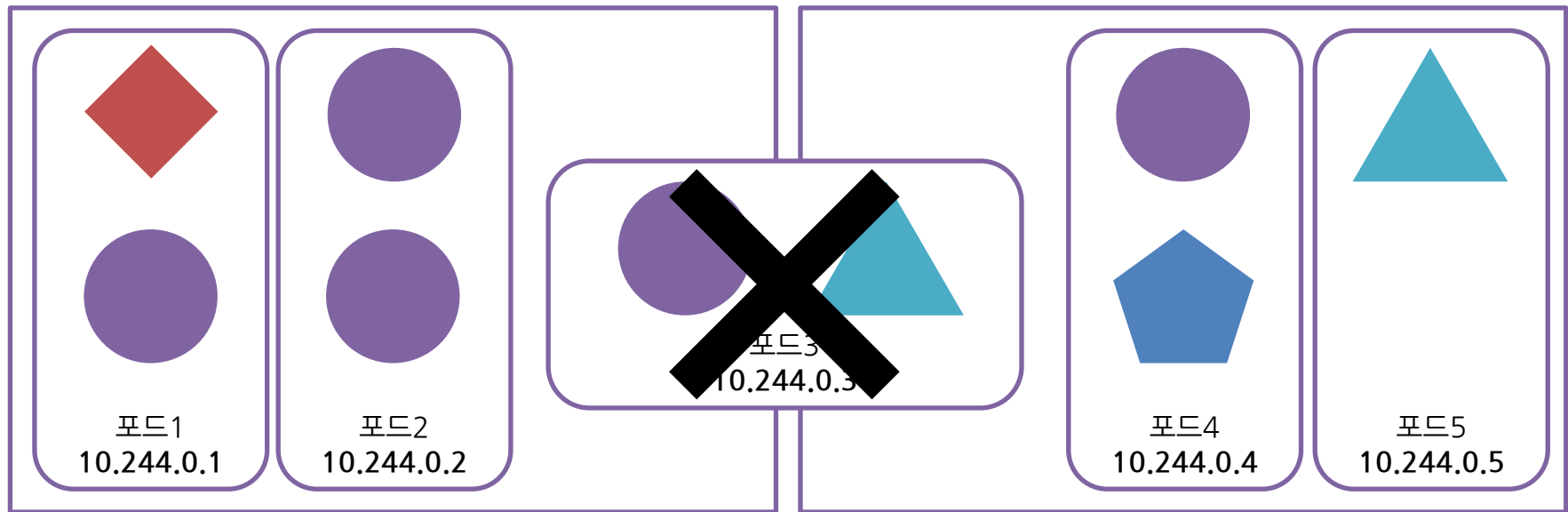




POD

▶ 포드

- 컨테이너의 공동 배포된 그룹이며 쿠버네티스의 기본 빌딩 블록을 대표
- 쿠버네티스는 컨테이너를 개별적으로 배포하지 않고 컨테이너의 포드를 항상 배포하고 운영
- 일반적으로 포드는 단일 컨테이너만 포함하지만 다수의 컨테이너를 포함 할 수 있음
- 포드는 다수의 노드에 생성되지 않고 단일 노드에서만 실행
- 여러 프로세스를 실행하기 위해서는 컨테이너 당 단일 프로세스가 적합
- 다수의 프로세스를 제어하려면? → 다수의 컨테이너를 다룰 수 있는 그룹이 필요!



▶▶ 포드의 관리

● 두 가지 장점

- 포드는 밀접하게 연관된 프로세스를 함께 실행하고 마치 하나의 환경에서 동작하는 것처럼 보임
- 그러나 동일한 환경을 제공하면서 다소 격리된 상태로 유지

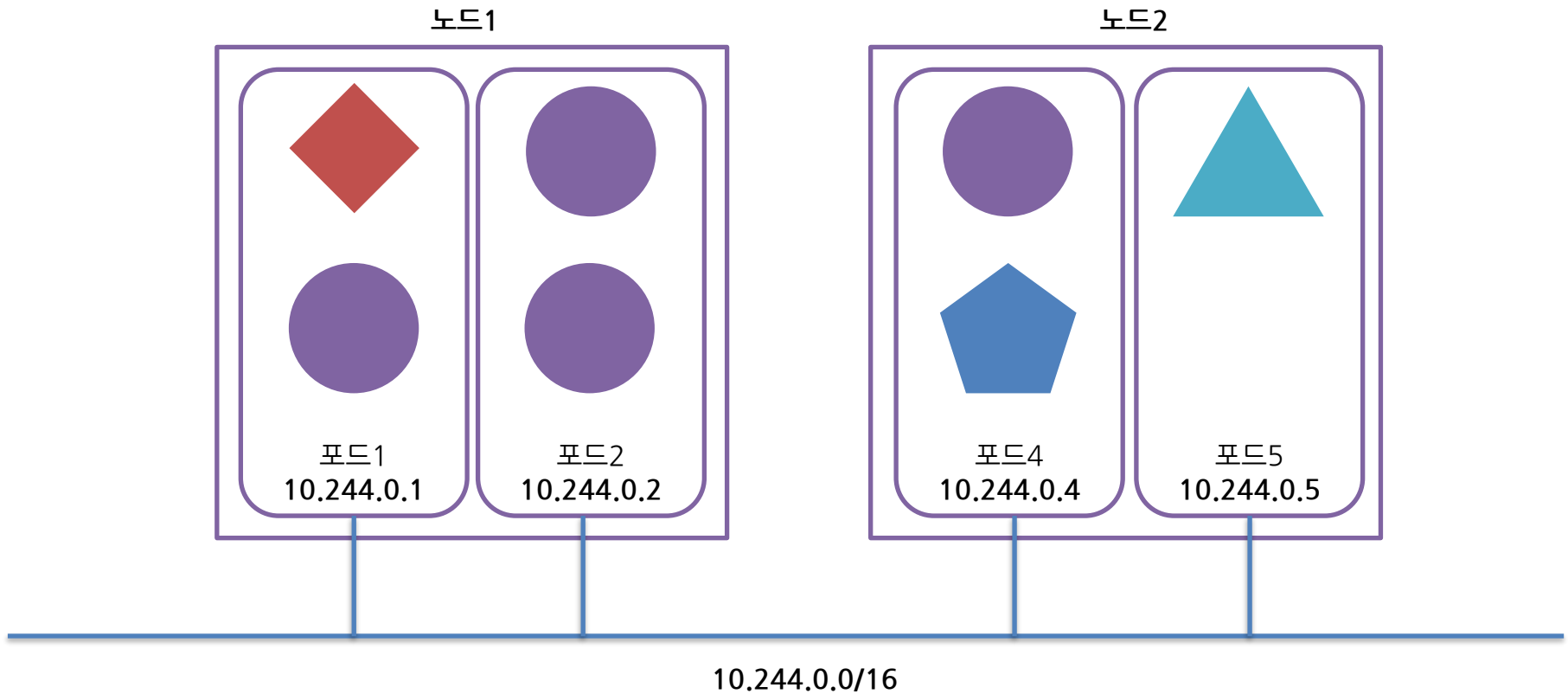
▶▶ 동일한 포드의 컨테이너 사이의 부분 격리

- 포드의 모든 컨테이너는 동일한 네트워크 및 UTS 네임스페이스에서 실행
- 같은 호스트 이름 및 네트워크 인터페이스를 공유 (포트 충돌 가능성 있음)
- 포드의 모든 컨테이너는 동일한 IPC 네임스페이스 아래에서 실행되며 IPC를 통해 통신 가능
 - 최신 쿠버네티스 및 도커 버전에서는 동일한 PID 네임스페이스를 공유할 수 있지만 이 기능은 기본적으로 활성화돼 있지 않다.

POD

▶ 플랫 인터 포드 네트워크 구조

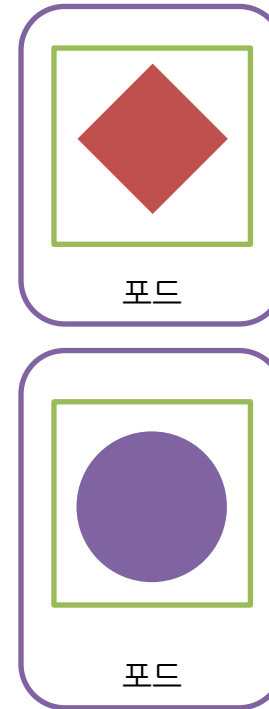
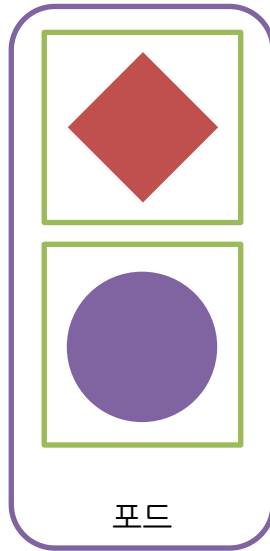
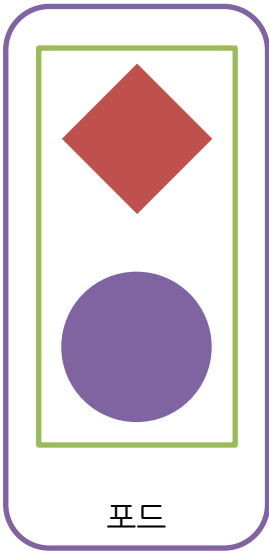
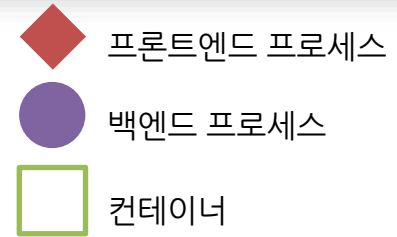
- 쿠버네티스 클러스터의 모든 포드는 공유된 단일 플랫, 네트워크 주소 공간에 위치
- 포드 사이에는 NAT 게이트웨이가 존재 하지 않음



POD

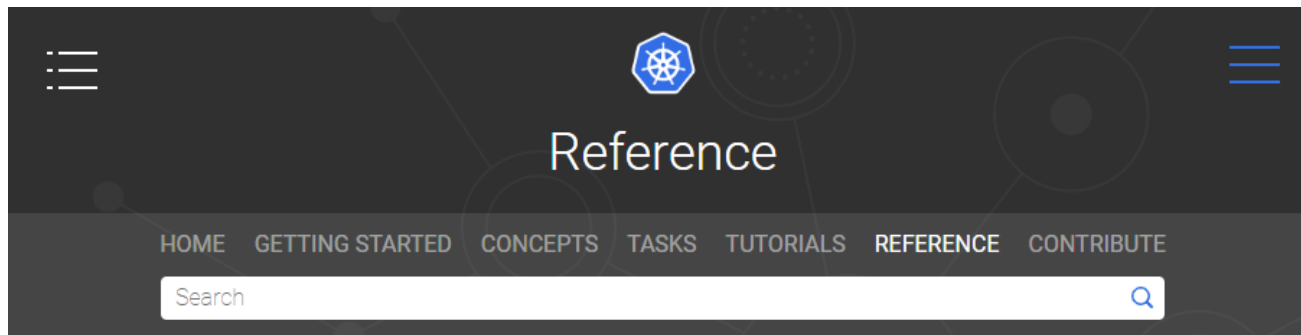
컨테이너를 포드 전체에 적절하게 구성하는 방법

- 다수의 포드로 멀티티어 애플리케이션 분할하기
- 각각 스케일링이 가능한 포드로 분할하기



YAML로 포드 디스크립터 만들기

- kubectl 실행 명령으로 간단한 리소스 작성 방법도 가능하지만 일부 항목에 대해서만 가능하며 저장에 용의하지 않음
- 모든 쿠버네티스 객체를 YAML로 정의하면 버전 제어 시스템에 저장 가능
- 모든 API에 대한 내용은 <http://kubernetes.io/docs/reference/> 참고



Reference



This section of the Kubernetes documentation contains references.

- [API Reference](#)
- [API Client Libraries](#)
- [CLI Reference](#)
- [Config Reference](#)
- [Design Docs](#)

▶▶ 포드 정의

- apiVersion, kind, 메타 데이터, 스펙, 스테이터스 로 구성
- 포드 정의 구성 요소
 - apiVersion: 쿠버네티스 api의 버전을 가리킴
 - kind: 어떤 리소스 유형인지 결정(포드 레플리카컨트롤러, 서비스 등)
 - 메타데이터: 포드와 관련된 이름, 네임스페이스, 레이블, 그 밖의 정보 존재
 - 스펙: 컨테이너, 볼륨 등의 정보
 - 상태: 포드의 상태, 각 컨테이너의 설명 및 상태, 포드 내부의 IP 및 그 밖의 기본 정보 등

POD

▶ 포드에서 YAML 파일 불러오기

```
$ kubectl get pod http-go -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-06-18T08:23:04Z"
  generateName: http-go-
  labels:
    run: http-go
  name: http-go-jvsg4
  namespace: default
  ownerReferences:
  - apiVersion: v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicationController
    name: http-go
    uid: 40b167be-91a2-11e9-8c69-0800278e464d
  resourceVersion: "8777"
  selfLink: /api/v1/namespaces/default/pods/http-go-jvsg4
  uid: 4b85d848-91a2-11e9-8c69-0800278e464d
```


POD

▶ 포드에서 YAML 파일 불러오기

```
# 포드의 상세 스펙(포드 컨테이너의 목록, 볼륨, 그 밖의 것들)
spec:
  containers:
  - image: gasbugs/http-go
    imagePullPolicy: Always
    name: http-go
    ports:
    - containerPort: 8080
      protocol: TCP
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-mts8j
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: slave2-virtualbox
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
[중략]
```

▶ 포드에서 YAML 파일 불러오기

포드의 컨테이너의 상세 상태

```
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2019-06-18T08:23:07Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2019-06-18T08:23:33Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2019-06-18T08:23:33Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2019-06-18T08:23:04Z"
    status: "True"
    type: PodScheduled
```

[중략]

POD

▶ 디스크립터 작성하기

- 간단히 다음과 같이 디스크립터 생성 (http-go-pod.yaml)

```
# 이 디스크립터는 쿠버네티스 API v1를 사용
apiVersion: v1
# 리소스 포드에 대한 설명
kind: Pod
metadata:
  # 포드의 이름
  name: http-go
spec:
  containers:
    # 생성할 컨테이너의 컨테이너 이미지
    - image: gasbugs/http-go
      name: http-go
      ports:
        # 응답 대기할 애플리케이션 포트
        - containerPort: 8080
          protocol: TCP
```

POD

kubectl에 디스크립터 작성 요령 확인 가능

```
$ kubectl explain pods
```

```
KIND: Pod
```

```
VERSION: v1
```

DESCRIPTION:

Pod is a collection of containers that can run on a host. This resource is created by clients and scheduled onto hosts.

FIELDS:

```
apiVersion      <string>
```

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info:
<https://git.k8s.io/community/contributors/devel/api-conventions.md#resources>

```
kind            <string>
```

Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info:
<https://git.k8s.io/community/contributors/devel/api-conventions.md#types-kinds>

POD

▶ 디스크립터를 사용해 포드 생성

- `kubectl create -f <파일명>`

```
$ kubectl create -f http-go-pod.yaml  
pod/ http-go created
```

▶ `kubectl log`로 포드의 로그 가져오기

- `kubectl logs http-go`

POD

▶ 컨테이너에서 호스트로 포트 포워딩

- 디버깅 혹은 다른 이유로 서비스를 거치지 않고 특정 포드와 통신하고 싶을 때 사용
- `kubectl port-forward` 명령으로 수행
- 컨테이너 8888 포트를 pod의 8080 포트로 전달

```
$ kubectl port-forward http-go 8080:8080
```

```
Forwarding from 127.0.0.1:8888 -> 8080
```

```
^Z
```

```
[1]+  Stopped                  kubectl port-forward http-go 8888:8080
```

```
$ bg
```

```
[1]+ kubectl port-forward http-go 8888:8080 &
```

```
$ curl 127.0.0.1:8888
```

```
Handling connection for 8888
```

```
Welcome! http-go
```

- 팁: 포트포워딩 종료는 프로세스 강제 kill로 수행한다.

POD

▶▶ 포드에 주석 추가하기

- 각 포드나 API 객체 설명이 추가
- 클러스터를 사용하는 모든 사람이 각 객체의 정보를 빠르게 확인 가능
- 예를 들어 객체를 만든 사람의 이름을 지정
- 공동 작업 가능
- 총 256KB까지 포함 가능

```
$ kubectl annotate pod http-go key="test1234"  
pod/http-go annotated
```

```
$ kubectl get pod http-go -o yaml
```

POD

▶ 포드 삭제

- 만든 포드 조회

```
$ kubectl get pod
```

NAME		READY	STATUS	RESTARTS	AGE
http-go	1/1	Running	0		7h12m

- 포드 삭제

- \$ kubectl delete pod <포드 이름>

- 전체 포드 삭제

- \$ kubectl delete pod --all

연습문제

- 모든 리소스 삭제
- YAML을 사용하여 도커이미지 jenkins로 Jenkins-manual 포드를 생성하기
- Jenkins 포드에서 curl 명령어로 로컬호스트:8080 접속하기
- Jenkins 포트를 8888로 포트포워딩하기
- 현재 Jenkins-manual의 설정을 yaml로 출력하기

▶ 라이브니스, 레디네스 프로브 구성

● Liveness, Readiness and Startup Probes

➤ Liveness Probe

- ✓ 컨테이너 살았는지 판단하고 다시 시작하는 기능
- ✓ 컨테이너의 상태를 스스로 판단하여 교착 상태에 빠진 컨테이너를 재시작
- ✓ 버그가 생겨도 높은 가용성을 보임

➤ Readiness Probe

- ✓ 포드가 준비된 상태에 있는지 확인하고 정상 서비스를 시작하는 기능
- ✓ 포드가 적절하게 준비되지 않은 경우 로드밸런싱을 하지 않음

➤ Startup Probe

- ✓ 애플리케이션의 시작 시기 확인하여 가용성을 높이는 기능
- ✓ Liveness와 Readiness의 기능을 비활성화

POD

▶ 라이브니스, 레디네스 프로브 구성

● Liveness 커맨드 설정 - 파일 존재 여부 확인

- 리눅스 환경 에서 커맨드 실행 성공 시 0 (컨테이너 유지)
- 실패하면 그 외 값 출력 (컨테이너 재시작)

exec-liveness.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
  livenessProbe:
    exec:
      command:
      - cat
      - /tmp/healthy
    initialDelaySeconds: 5
    periodSeconds: 5
```

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

POD

▶ 라이브니스, 레디네스 프로브 구성

● Liveness 웹 설정 - http 요청 확인

- 서버 응답 코드가 200이상 400미만 (컨테이너 유지)
- 서버 응답 코드가 그 외 (컨테이너 재시작)

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

http-liveness.yaml

POD

▶ 라이브니스, 레디네스 프로브 구성

● Readiness TCP 설정

- 준비 프로브는 8080포트를 검사
- 5초 후부터 검사 시작
- 검사 주기는 10초
- → 서비스를 시작해도 된다!

● Liveness TCP 설정

- 활성화 프로브는 8080포트를 검사
- 15초 후부터 검사 시작
- 검사 주기는 20초
- → 컨테이너를 재시작하지 않아도 된다!

소스코드 출처:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>

```
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

tcp-liveness-readiness.yaml

▶ 라이브니스, 레디네스 프로브 구성

● Statup Probe

- 시작할 때까지 검사를 수행
- http 요청을 통해 검사
- 30번을 검사하며 10초 간격으로 수행
- 300(30*10)초 후에도 포드가 정상 동작하지 않는 경우 종료
- → 300초 동안 포드가 정상 실행되는 시간을 벌어줌

Startup Probe 예제

```
ports:
- name: liveness-port
  containerPort: 8080
  hostPort: 8080

livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 10

startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

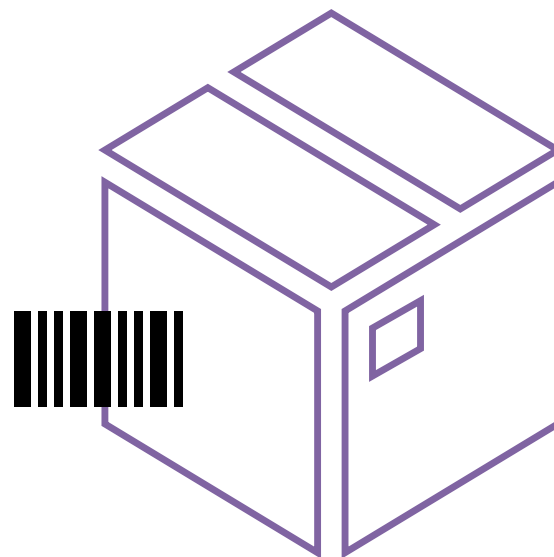
레이블과 셀렉터

레이블과 셀렉터

레이블을 이용한 포드 구성

● 레이블이란?

- 모든 리소스를 구성하는 매우 간단하면서도 강력한 쿠버네티스 기능
- 리소스에 첨부하는 임의의 키/값 쌍(예 app: test)
- 레이블 셀렉터를 사용하면 각종 리소스를 필터링하여 선택할 수 있음
- 리소스는 한 개 이상의 레이블을 가질 수 있음
- 리소스를 만드는 시점에 레이블을 첨부
- 기존 리소스에도 레이블의 값을 수정 및 추가 가능
- 모든 사람이 쉽게 이해할 수 있는 체계적인 시스템을 구축 가능
 - ✓ app: 애플리케이션 구성요소, 마이크로서비스 유형 지정
 - ✓ rel: 애플리케이션의 버전 지정

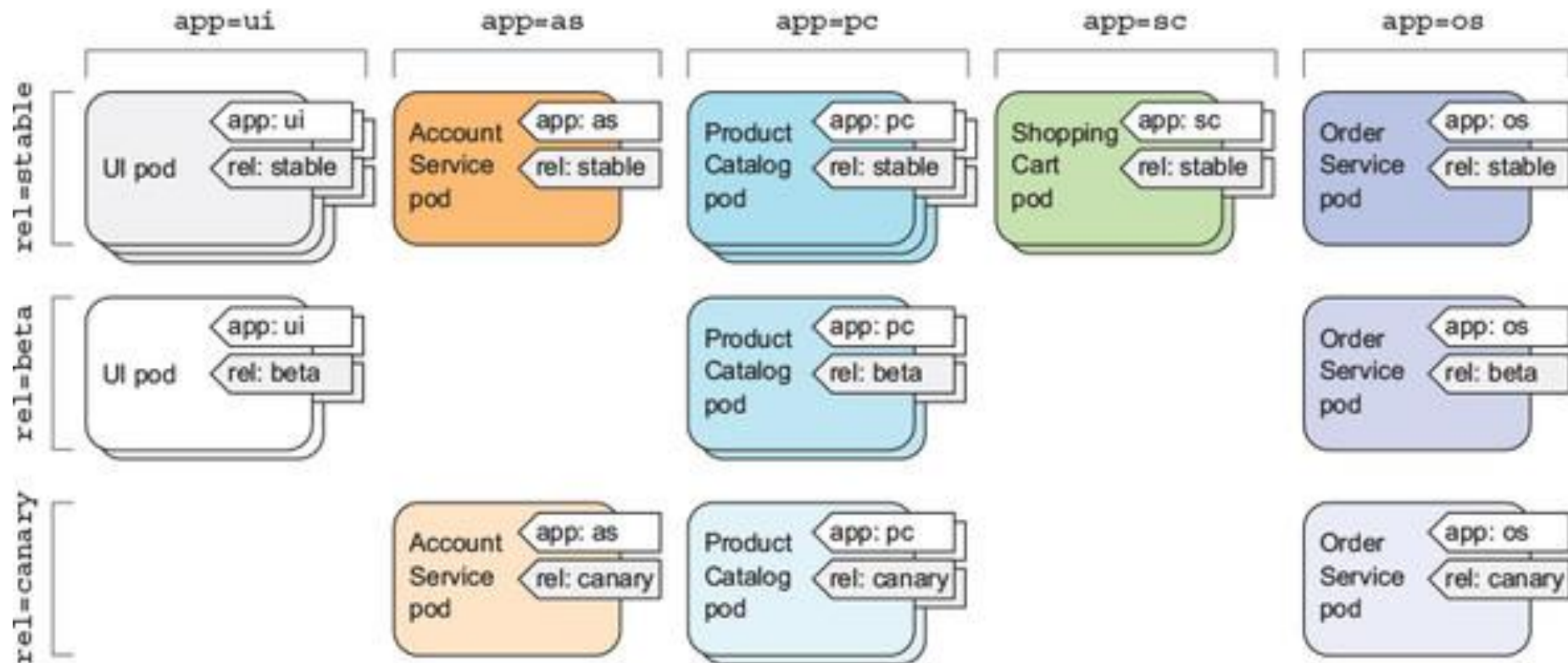


포드를 인식하기 위한 바코드

레이블과 셀렉터

레이블을 이용한 포드 구성

- <https://livebook.manning.com/#!/book/kubernetes-in-action/chapter-3/141>



레이블과 셀렉터

▶ 포드 생성 시 레이블을 지정하는 방법

```
$ kubectl create -f http-go-pod-v2.yaml  
pod/http-go-v2 created
```

http-go-pod-v2.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: http-go-v2  
  labels:  
    creation_method: manual  
    env: prod  
spec:  
  containers:  
  - image: gasbugs/http-go  
    name: http-go  
    ports:  
    - containerPort: 8080  
      protocol: TCP
```

레이블과 셀렉터

▶ 레이블을 추가 및 수정하는 방법

- 새로운 레이블을 추가할 때는 label 명령어를 사용

```
$ kubectl label pod http-go-v2 test=foo
pod/http-go-v2 labeled
```

- 기존의 레이블을 수정할 때는 --overwrite 옵션을 주어서 실행

```
$ kubectl label pod http-go-v2 rel=beta
error: 'rel' already has a value (canary), and --overwrite is false
```

```
$ kubectl label pod http-go-v2 rel=beta --overwrite
pod/http-go-v2 labeled
```

- 레이블 삭제

```
$ kubectl label pod http-go-v2 rel-
```

레이블과 셀렉터

▶ 레이블 확인하기

● 레이블 보여주기

```
$ kubectl get pod --show-labels
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
http-go	1/1	Running	0	53m	<none>
http-go-v2	1/1	Running	0	2m5s	app=http-go,foo=bar,rel=beta,test=foo

● 특정 레이블 컬럼으로 확인

```
$ kubectl get pod -L app,rel
```

NAME	READY	STATUS	RESTARTS	AGE	APP	REL
http-go	1/1	Running	0	62m		
http-go-v2	1/1	Running	0	11m	http-go	beta

레이블과 셀렉터

레이블로 필터링하여 검색

```
$ kubectl get pod --show-labels -l 'env'
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
http-go-v2	1/1	Running	0	3m10s	creation_method=manual,env=prod,rel=beta,test=foo

```
$ kubectl get pod --show-labels -l '!env'
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
http-go	1/1	Running	0	2m	<none>

```
$ kubectl get pod --show-labels -l 'env!=test'
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
http-go	1/1	Running	0	4m	<none>
http-go-v2	1/1	Running	0	5m14s	creation_method=manual,env=prod,rel=beta,test=foo

```
$ kubectl get pod --show-labels -l 'env!=test,rel=beta'
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
http-go-v2	1/1	Running	0	5m58s	creation_method=manual,env=prod,rel=beta,test=foo

레이블과 셀렉터

레이블 배치 전략

- <https://www.replex.io/blog/9-best-practices-and-examples-for-working-with-kubernetes-labels?fbclid=IwAR0S2tT3iw8FlkVYWwyjL8OW6IWigXfk0fDkAk57o6re1rRnoSRRzFVXiM>

[KUBERNETES]

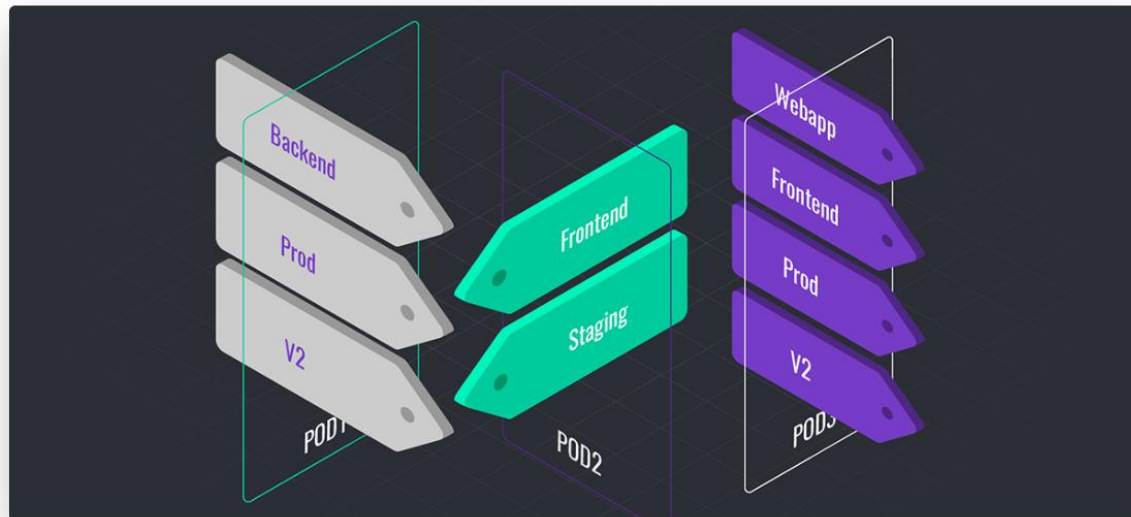
9 Best Practices and Examples for Working with Kubernetes Labels

Kubernetes labels allow DevOps teams to identify, select and operate on Kubernetes objects. In this blog post we outline 9 best practices for working with Kubernetes Labels and examples for Kubernetes label keys and values



HASHAM HAIDER

September 28, 2018 | 11 minute read



레이블과 셀렉터

레이블 배치 전략

- 확장 가능한 쿠버네티스 레이블 예제

레이블 키	설명	레이블 값
Application-ID/Application-name	응용 프로그램 이름 또는 ID	my-awesome-app/app-nr-2345
Version-nr	버전 번호	ver-0.9
Owner	개체가 속한 팀 또는 개인	Team-kube/Josh
Stage/Phase	개발 단계 또는 위치	Dev, staging, QA, Canary, Production
Release-nr	릴리즈 번호	release-nr-2.0.1
Tier	앱이 속한 계층	front-end/back-end
Customer-facing	고객에게 직접 서비스 하는 앱 여부	Yes/No
App-role	앱의 역할	Cache/Web/Database/Auth
Project-ID	연관된 프로젝트 ID	my-project-276
Customer-ID	자원을 할당한 고객 ID	customer-id-29

레이블과 셀렉터

▶▶ 연습문제

- YAML 파일을 사용하여 app=nginx 레이블을 가진 포드를 생성하라.
- app=nginx를 가진 포드를 get하라.
- get된 포드의 레이블의 app을 확인하라.
- app=nginx 레이블을 가진 포드에 team=dev1 레이블을 추가하라.



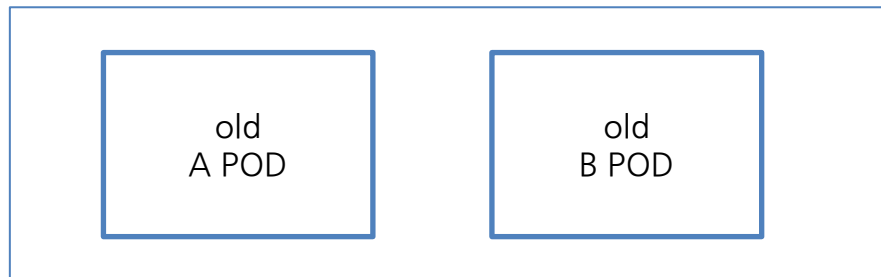
ReplicaSet

ReplicaSet

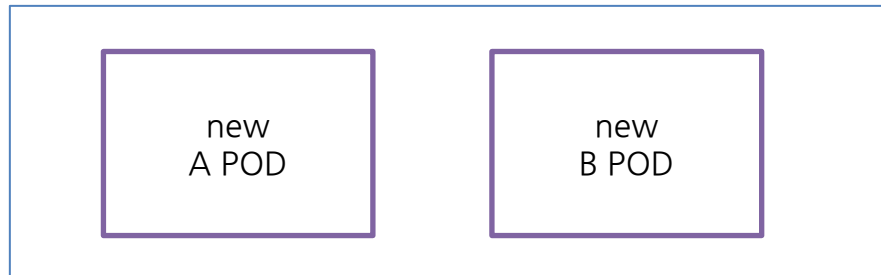
▶ 레플리케이션컨트롤러

- 포드가 항상 실행되도록 유지하는 쿠버네티스 리소스
- 노드가 클러스터에서 사라지는 경우 해당 포드를 감지하고 대체 포드 생성
- 실행 중인 포드의 목록을 지속적으로 모니터링으로 하고 '유형'의 실제 포드 수가 원하는 수와 항상 일치하는지 확인

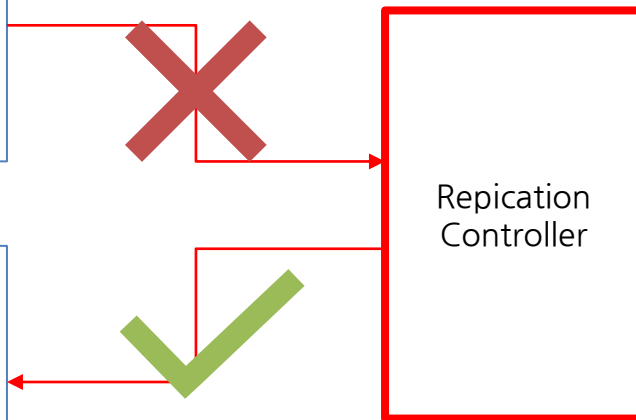
워커 노드1



워커 노드2



장애 발생



레플리케이션(Replication): 데이터 저장과 백업하는 방법과 관련이 있는 데이터를 호스트 컴퓨터에서 다른 컴퓨터로 복사하는 것 (위키백과)

ReplicaSet

▶ 레플리케이션컨트롤러의 세 가지 요소

- 레플리케이션컨트롤러가 관리하는 포드 범위를 결정 하는 레이블 선택터
- 실행해야 하는 포드의 수를 결정하는 복제본 수
- 새로운 포드의 모양을 설명하는 포드 템플릿

▶ 레플리케이션컨트롤러의 장점

- 포드가 없는 경우 새 포드를 항상 실행
- 노드에 장애 발생 시 다른 노드에 복제본 생성
- 수동, 자동으로 수평 스케일링

ReplicaSet

▶ 레플리케이션컨트롤러의 YAML 작성

- [http-go-rc.yaml](#)

실행해야 하는 포드의 수를 결정하는 복제본 수

레플리케이션컨트롤러가 관리하는 포드 범위를 결정 하는 레이블 선택터

새로운 포드의 모양을 설명하는 포드 템플릿

```
apiVersion: v1
kind: ReplicationController
metadata:
  # 레플리케이션컨트롤러 이름
  name: rc-nodejs
spec:
  # 복제본 수
  replicas: 3
  # 라벨 선택터
  selector:
    app: nodejs

# 포드 템플릿
template:
  metadata: 포드와 완전히 동일
    labels:
      app: nodejs
  spec:
    containers:
      - name: nodejs
        image: gasbugs/nodejs
        ports:
          - containerPort: 8080
```

ReplicaSet

▶ 레플리케이션컨트롤러의 YAML 작성

- 실행 중인 레플리케이션컨트롤러와 포드 확인

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
rc-http-go-htktl	1/1	Running	0	23s
rc-http-go-lvb25	1/1	Running	0	23s
rc-http-go-znmlz	1/1	Running	0	23s

```
$ kubectl get rc
```

NAME	DESIRED	CURRENT	READY	AGE
rc-http-go	3	3	3	4m23s

ReplicaSet

▶ 레플리케이션컨트롤러의 YAML 작성

- 포드를 임의로 정지시켜 반응 확인

```
$ kubectl delete pod rc-http-go-znmlz  
pod "rc-http-go-znmlz" deleted
```

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
rc-http-go-htktl	1/1	Running	0	6m44s
rc-http-go-knfzp	1/1	Running	0	39s
rc-http-go-lvb25	1/1	Running	0	6m44s

ReplicaSet

▶ 레플리케이션 정보 확인

```
$ kubectl describe rc rc-http-go
```

Name: rc-http-go

Namespace: default

Selector: app=http-go

Labels: app=http-go

Annotations: <none>

Replicas: 3 current / 3 desired

Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed

Pod Template:

Labels: app=http-go

Containers:

http-go:

Image: gasbugs/http-go

Port: 8080/TCP

[중략]

Volumes: <none>

Events:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	SuccessfulCreate	7m47s	replication-controller	Created pod: rc-http-go-lvb25
Normal	SuccessfulCreate	7m47s	replication-controller	Created pod: rc-http-go-znmlz
Normal	SuccessfulCreate	7m47s	replication-controller	Created pod: rc-http-go-htktl
Normal	SuccessfulCreate	102s	replication-controller	Created pod: rc-http-go-knfzp

ReplicaSet

🚦 노드 통신 직접 다운시켜 보기

```
$ gcloud compute ssh gke-standard-cluster-1-default-pool-b1e2cd6b-z2nd  
$ sudo ifconfig eth0 down  
(응답없음...)
```

```
$ kubectl get node
```

NAME	STATUS	ROLES	AGE	VERSION
gke-standard-cluster-1-default-pool-b1e2cd6b-3r3q	Ready	<none>	10h	v1.12.8-gke.6
gke-standard-cluster-1-default-pool-b1e2cd6b-cb83	Ready	<none>	10h	v1.12.8-gke.6
gke-standard-cluster-1-default-pool-b1e2cd6b-z2nd	NotReady	gpu	10h	v1.12.8-gke.6

ReplicaSet

▶ 레플리카컨트롤러의 관리 레이블 벗어나기

```
$ kubectl get pod -L app
```

NAME	READY	STATUS	RESTARTS	AGE	APP
rc-http-go-p65gc	1/1	Running	0	108s	http-go
rc-http-go-wtltz	1/1	Running	0	108s	http-go
rc-http-go-zhdj5	1/1	Running	0	108s	http-go

```
$ kubectl label pod rc-http-go-zhdj5 app=http-go2 --overwrite  
pod/rc-http-go-zhdj5 labeled
```

```
$ kubectl get pod -L app
```

NAME	READY	STATUS	RESTARTS	AGE	APP
rc-http-go-8x9w7	0/1	ContainerCreating	0	4s	http-go
rc-http-go-p65gc	1/1	Running	0	4m47s	http-go
rc-http-go-wtltz	1/1	Running	1	4m47s	http-go
rc-http-go-zhdj5	1/1	Running	0	4m47s	http-go2

- 포드의 레이블이 변경되어 관리 밖으로 벗어나면 이를 건드리지 않고 새로운 포드를 생성

ReplicaSet

▶ 레플리케이션컨트롤러 설정 바꾸기

- 다음 명령어로 설정파일 접근

```
$ kubectl edit rc rc-http-go
```

- vim이 열리면 replicas 개수를 3에서 20개로 수정하고 저장 종료

- 포드의 수 변화 확인

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
rc-http-go-4mfdg	1/1	Running	0	16s
rc-http-go-6cgvh	1/1	Running	0	16s
rc-http-go-8g9h8	1/1	Running	0	16s
rc-http-go-8x9w7	1/1	Running	0	5m12s
rc-http-go-9sgzn	0/1	Pending	0	16s
rc-http-go-b4t4z	0/1	ContainerCreating	0	16s
rc-http-go-ctrcn	1/1	Running	0	16s
rc-http-go-fvd96	0/1	Pending	0	16s
rc-http-go-hgxnd	0/1	Pending	0	15s
rc-http-go-jkqlr	0/1	Pending	0	16s
<...>				

ReplicaSet

▶ 레플리케이션컨트롤러 설정 바꾸기

- 다음 명령어로 레플리케이션컨트롤러 설정 변경

```
$ kubectl scale rc rc-http-go --replicas=10
replicationcontroller/rc-http-go scaled
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
rc-http-go-4mfdg	1/1	Running	0	9m51s
rc-http-go-6cgvh	1/1	Running	0	9m51s
rc-http-go-8g9h8	1/1	Running	0	9m51s
rc-http-go-8x9w7	1/1	Running	0	14m
rc-http-go-b4t4z	1/1	Terminating	0	9m51s
rc-http-go-ctrcn	1/1	Running	0	9m51s
rc-http-go-p65gc	1/1	Running	0	19m
rc-http-go-wtltz	1/1	Running	1	19m
rc-http-go-x9vj5	1/1	Running	0	9m51s
rc-http-go-z2gc6	1/1	Running	0	9m51s
rc-http-go-z6ctk	1/1	Terminating	0	9m51s
rc-http-go-zhdj5	1/1	Running	0	19m
rc-http-go-zt4dx	1/1	Running	0	9m51s

ReplicaSet

▶ 레플리케이션컨트롤러 삭제 바꾸기

- 일반적인 삭제 명령어와 동일

```
$ kubectl delete rc rc-http-go  
replicationcontroller "rc-http-go" deleted
```

- 실행 시키고 있는 포드는 계속 실행을 유지하고 싶은 경우에는 --cascade 옵션 사용

```
$ kubectl delete rc rc-http-go --cascade=false  
replicationcontroller "rc-http-go" deleted
```

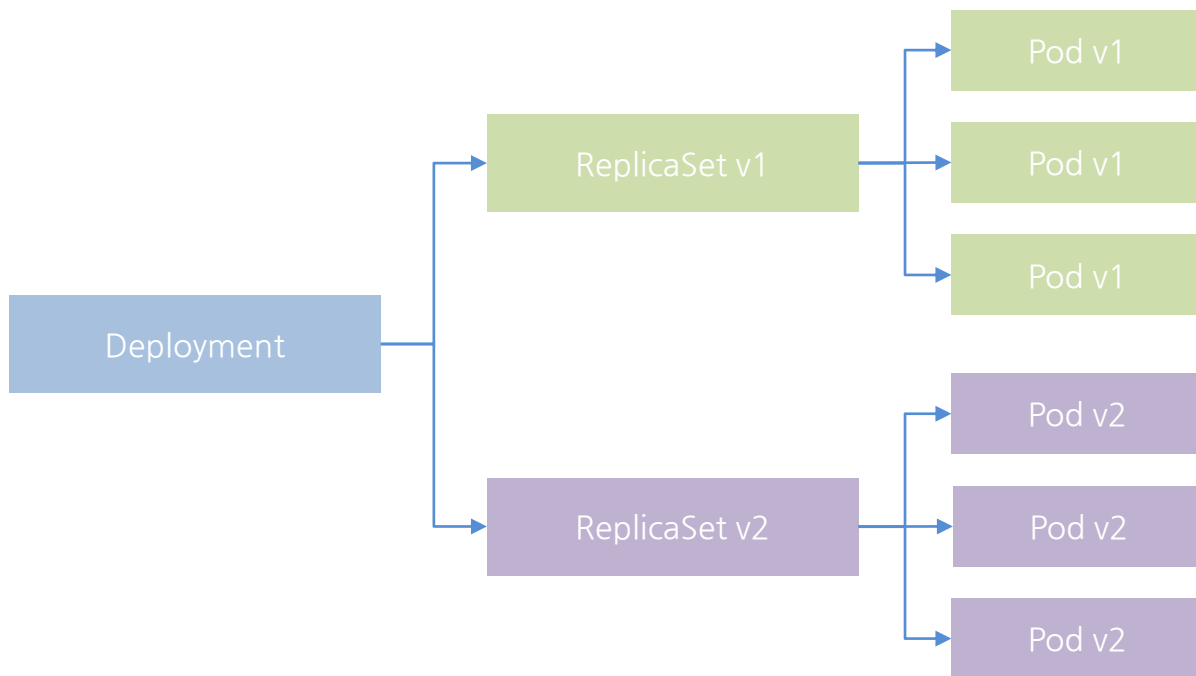
```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
rc-http-go-crllr	1/1	Running	0	29s
rc-http-go-cvqsb	1/1	Running	0	29s
rc-http-go-vnvsvd	1/1	Running	0	29s

ReplicaSet

▶ 레플리카셋의 등장

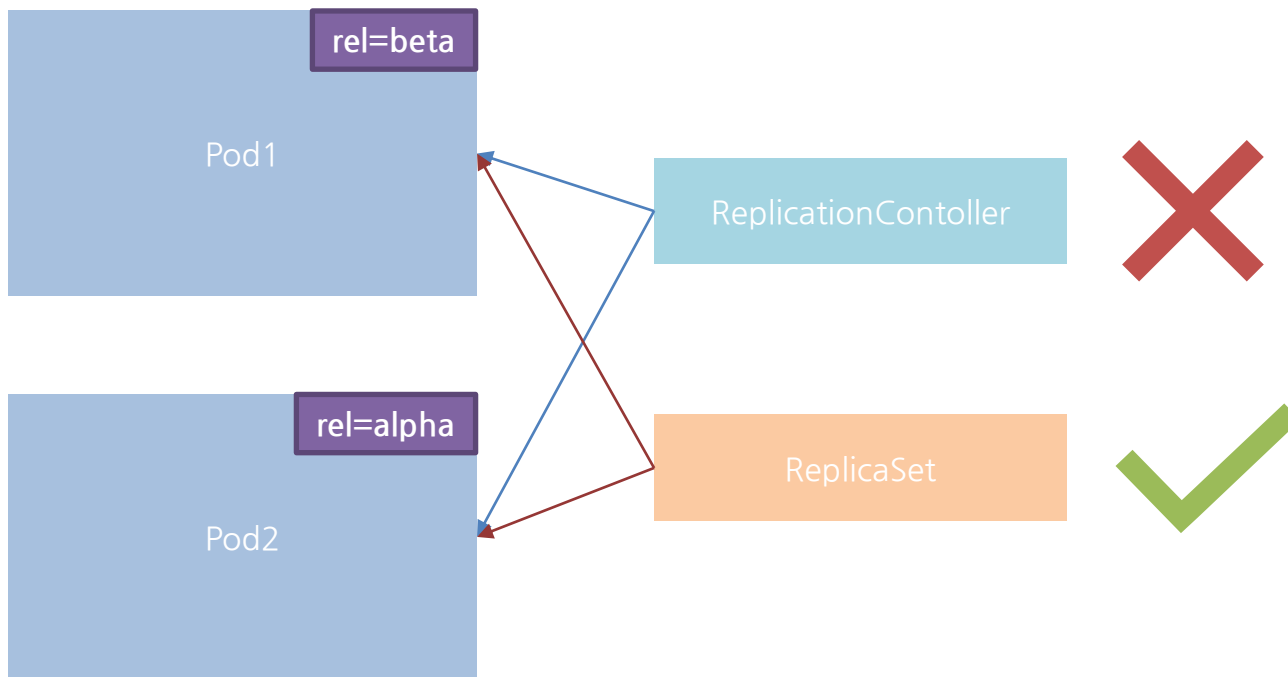
- 쿠버네티스 1.8 버전부터 디플로이먼트 (Deployment), 데몬셋 (Daemonset), 레플리카셋 (ReplicaSet), 스테이트풀셋 (StatefulSet) 네 API가 베타로 업데이트되고 1.9 버전에서는 정식 버전으로 업데이트됨
- 레플리카셋은 차세대 레플리케이션컨트롤러로 레플리케이션컨트롤러를 완전히 대체 가능
- 초기 쿠버네티스에서 제공했기 때문에 현장에서는 여전히 계속 사용중인 경우 존재



ReplicaSet

▶ 레플리케이션컨트롤러 vs. 레플리카셋

- 레플리카셋과 레플리케이션컨트롤러는 거의 동일하게 동작
- 레플리카셋이 더 풍부한 표현식 포드 셀렉터 사용 가능
 - 레플리케이션컨트롤러: 특정 레이블을 포함하는 포드가 일치하는지 확인
 - 레플리카셋: 특정 레이블이 없거나 해당 값과 관계없이 특정 레이블 키를 포함하는 포드를 매치하는지 확인



ReplicaSet

▶ 레플리카셋 생성

- 대부분의 요소는 거의 비슷
- apiVersion: **apps/v1beta2**
- kind: **ReplicaSet**
- matchExpressions: 레이블을 매칭하는 별도의 표현 방식 존재

```
Selector: # This is our label selector field.  
matchLabels:  
  tier: some-Tier  
matchExpressions:  
  - {key: tier, operator: In, values: [some-Tier]} #
```

```
selector:  
  matchLabels:  
    app: http-go
```

```
$ kubectl get pod --show-labels
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
http-go-rs-45mmq	1/1	Running	0	50s	app=http-go
http-go-rs-ddrfm	1/1	Running	0	50s	app=http-go
http-go-rs-wxvbf	1/1	Running	0	50s	app=http-go

```
apiVersion: apps/v1beta2  
kind: Replicaset  
metadata:  
  name: http-go  
spec:  
  replicas: 3  
  selector:  
    matchExpressions:  
      - key: app  
        operator: In  
        values:  
          - http-go  
  template:  
    metadata:  
      labels:  
        app: http-go  
    spec:  
      containers:  
        - name: http-go  
          image: gasbugs/http-go
```

ReplicaSet

▶ 레플리카셋의 조회와 삭제

- 다른 리소스와 모두 동일
- 조회: `$ kubectl get rs`
- 상세조회: `$ kubectl describe rs http-go-rs`
- 삭제 `$ kubectl delete rs http-go-rs`

ReplicaSet

▶▶ 연습문제

- nginx를 3개 생성하는 rs-nginx 레플리카셋을 생성하라.
- rs-nginx 포드의 개수를 10개로 스케일링하라.

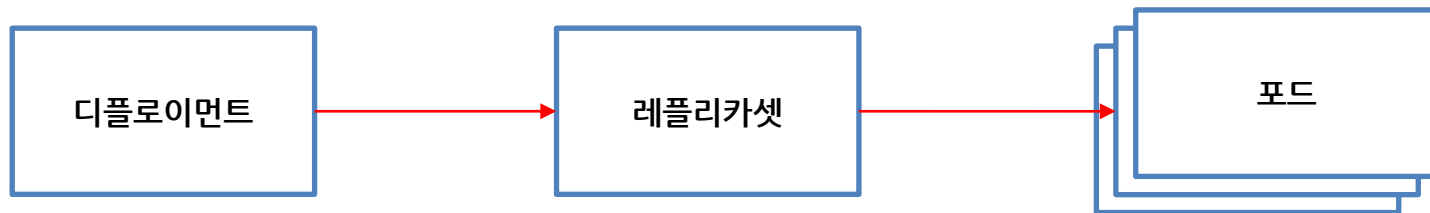


Deployment

Deployment

디플로이먼트

- 애플리케이션을 다운 타임 없이 업데이트 가능하도록 도와주는 리소스!
- 레플리카셋과 레플리케이션컨트롤러 상위에 배포되는 리소스



- 모든 포드를 업데이트하는 방법
 - 잠깐의 다운 타임 발생 (새로운 포드를 실행시키고 작업이 완료되면 오래된 포드를 삭제)
 - 롤링 업데이트

Deployment

디플로이먼트 작성 요령

- 포드의 metadata 부분과 spec 부분을 그대로 옮김
- Deployment의 spec.template에는 배포할 포드를 설정
- replicas에는 이 포드를 몇 개를 배포할지 명시
- label은 디플로이먼트가 배포한 포드를 관리하는데 사용됨

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
```



Deployment

▶ 디플로이먼트 스케일링

- `kubectl edit deploy <deploy name>` 명령을 사용해 yaml 파일을 직접 수정하여 replicas 수를 조정
- `kubectl scale deploy <deploy name> --replicas=<number>` 명령을 사용해 replicas 수 조정

Deployment

연습문제

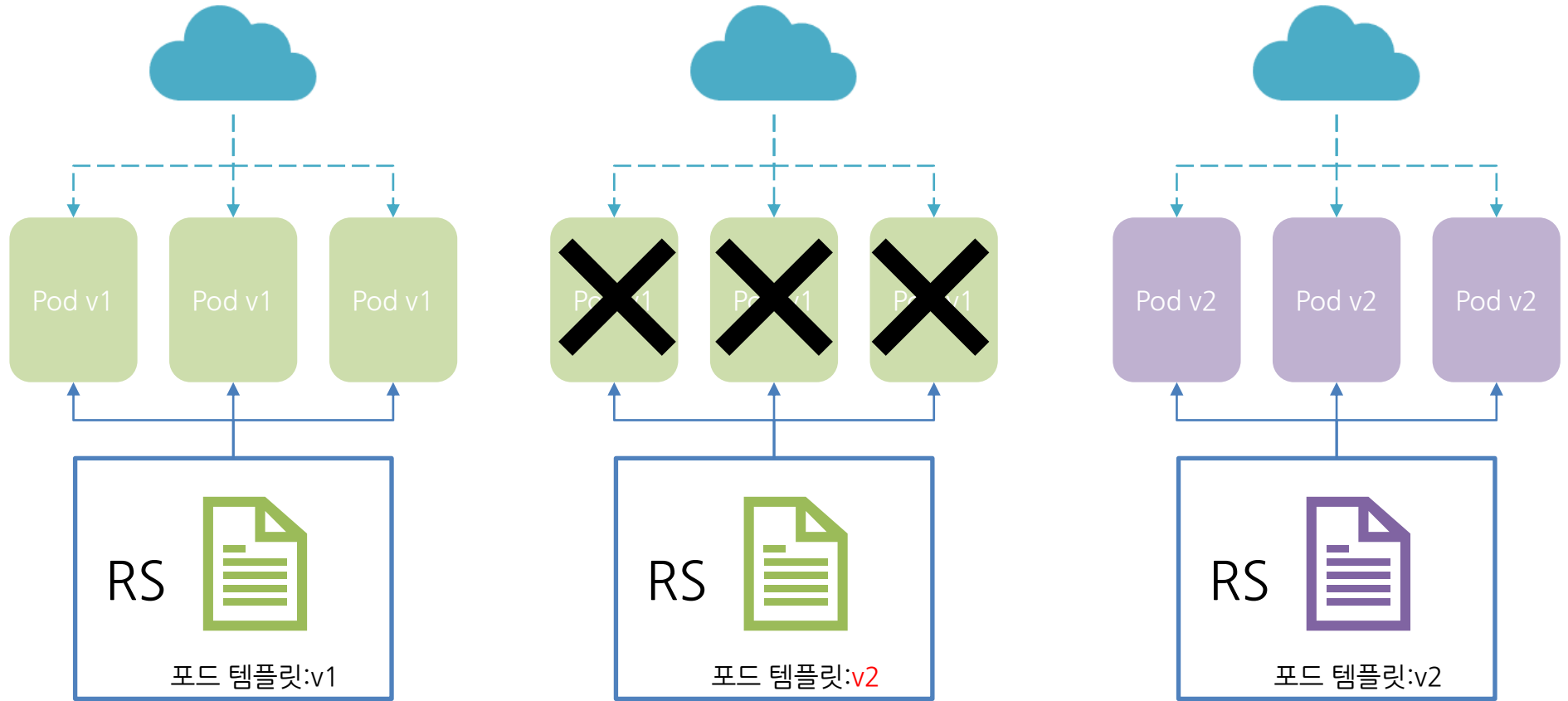
- jenkins 디플로이먼트를 deploy-jenkins를 생성하라.
- jenkins 디플로이먼트로 배포되는 앱을 app: jenkins-test로 레이블링하라.
- 디플로이먼트로 배포된 포드를 하나 삭제하고 이후 생성되는 포드를 관찰하라.
- 새로 생성된 포드의 레이블을 바꾸어 Deployment의 관리 영역에서 벗어나게 하라.
- Scale 명령을 사용해 레플리카 수를 5개로 정의한다.
- edit 기능을 사용하여 10로 스케일링하라.

애플리케이션 롤링 업데이트와 롤백

애플리케이션 롤링 업데이트와 롤백

기존 모든 포드를 삭제 후 새로운 포드 생성

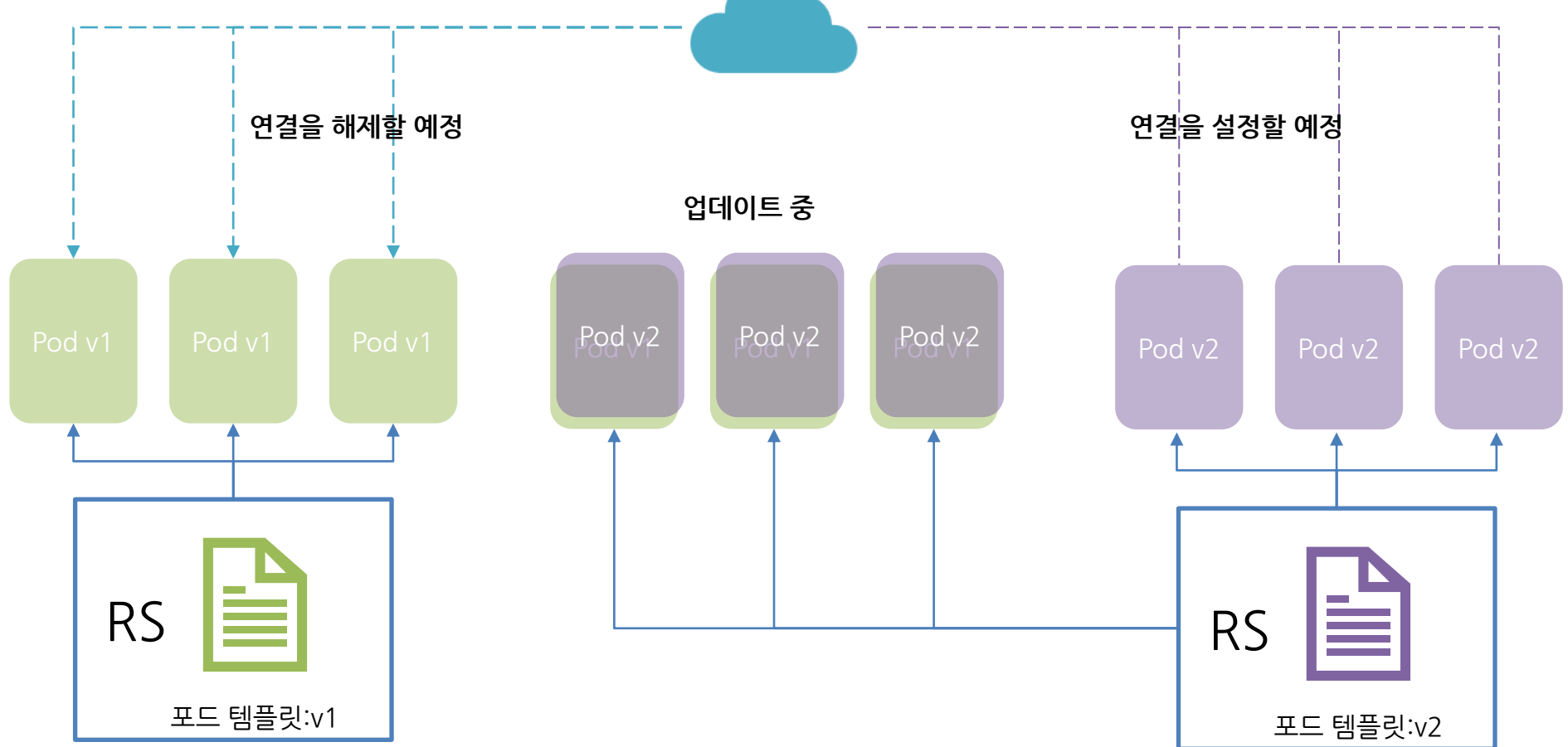
- 잠간의 다운 타임 발생



애플리케이션 롤링 업데이트와 롤백

새로운 포드를 실행시키고 작업이 완료되면 오래된 포드를 삭제

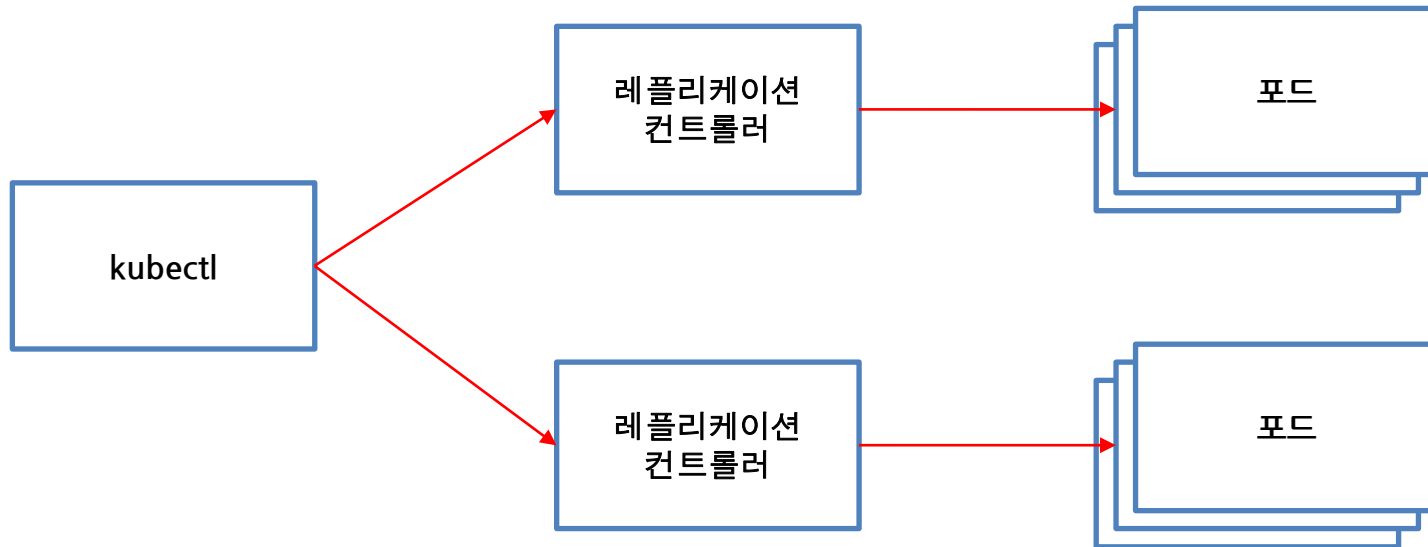
- 새 버전을 실행하는 동안 구 버전 포드와 연결
- 서비스의 레이블셀렉터를 수정하여 간단하게 수정가능



애플리케이션 롤링 업데이트와 롤백

▶ 레플리케이션컨트롤러가 제공하는 롤링 업데이트

- 이전에는 kubectl을 사용해 스케일링을 사용하여 수동으로 롤링 업데이트 진행 가능
- Kubectl 중단되면 업데이트는 어떻게 될까?
- 레플리케이션컨트롤러 또는 레플리카셋을 통제할 수 있는 시스템이 필요



애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 생성

- 레이블 선택터, 원하는 복제본 수, 포트 템플릿
- 디플로이먼트의 전략을 yaml에 지정하여 사용 가능
- 먼저 업데이트 시나리오를 위해 3개의 도커 이미지를 준비
 - gasbugs/http-go:v1
 - gasbugs/http-go:v2
 - gasbugs/http-go:v3

dockerfiles

```
FROM golang:1.11
WORKDIR /usr/src/app
COPY main /usr/src/app
CMD ["/usr/src/app/main"]
```

```
package main
```

main.go

```
import (  
    "fmt"  
    "github.com/julienschmidt/httprouter"  
    "net/http"  
    "log"  
)  
  
func Index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {  
    fmt.Fprint(w, "Welcome! v1\n")  
}  
  
func main() {  
    router := httprouter.New()  
    router.GET("/", Index)  
  
    log.Fatal(http.ListenAndServe(":8080", router))  
}
```

애플리케이션 롤링 업데이트와 롤백

디플로이먼트 생성 YAML 만들기

- 버전을 이름에 넣을 필요가 없음
(업데이트 되어도 동일한 디플로이먼트를 사용)

```
$ kubectl create -f http-go-deployment.yaml --record=true
deployment.apps/http-go created
```

```
$ kubectl get deployment
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
http-go	3	3	3	3	6m15s

```
$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
http-go-6f8b8f95db	3	3	3	6m2s

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
http-go-6f8b8f95db-j88dn	1/1	Running	0	6m6s
http-go-6f8b8f95db-n6gdt	1/1	Running	0	6m6s
http-go-6f8b8f95db-z8pg4	1/1	Running	0	6m6s

```
$ kubectl rollout status deployment http-go
deployment "http-go" successfully rolled out
```

rollout을 통해서도 상태 확인 가능

http-go-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: http-go-deployment
  labels:
    app: http-go
spec:
  replicas: 3
  template:
    metadata:
      name: http-go
      labels:
        app: http-go
    spec:
      containers:
        - image: gasbugs/http-go:v1
          name: http-go
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 전략(StrategyType)

● Rolling Update(기본값)

- 오래된 포드를 하나씩 제거하는 동시에 새로운 포드 추가
- 요청을 처리할 수 있는 양은 그대로 유지
- 반드시 이전 버전과 새 버전을 동시에 처리 가능하도록 설계한 경우에만 사용해야 함

● Recreate

- 새 포드를 만들기 전에 이전 포드를 모두 삭제
- 여러 버전을 동시에 실행 불가능
- 잠깐의 다운 타임 존재

```
spec:  
  strategy:  
    type: RollingUpdate
```

● 업데이트 과정을 보기 위해 업데이트 속도 조절

```
$ kubectl patch deployment http-go -p '{"spec": {"minReadySeconds": 10}}'  
deployment.extensions/http-go patched
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 실행 준비

- 디플로이먼트를 모니터링하는 프로그램 실행

```
$ while true; curl <ip>; sleep 1; done  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
...
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 실행

- 새로운 터미널을 열어 이미지 업데이트 실행

```
$ kubectl set image deployment http-go http-go=gasbugs/http-go:v2  
deployment.extensions/http-go image updated
```

- 모니터링하는 시스템에서 관찰

```
$ while true; curl <ip>; sleep 1; done  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
Welcome! http-go:v1  
...  
Welcome! http-go:v1  
Welcome! http-go:v2  
Welcome! http-go:v2  
Welcome! http-go:v1
```

애플리케이션 롤링 업데이트와 롤백

▶ 디플로이먼트 업데이트 실행 결과

- 업데이트한 이력을 확인

- 리비전의 개수는 디폴트로 10개까지 저장

```
$ kubectl rollout history deployment http-go  
deployment.extensions/http-go
```

```
REVISION  CHANGE-CAUSE
```

```
1          <none>
```

```
2          kubectl set image deployment http-go http-go=gasbugs/http-go:v2
```


애플리케이션 롤링 업데이트와 롤백

▶▶ 롤백 실행하기

- 롤백을 실행하면 이전 업데이트 상태로 돌아감
- 롤백을 하여도 히스토리의 리비전 상태는 이전 상태로 돌아가지 않음

```
$ kubectl set image deployment http-go http-go=gasbugs/http-go:v3  
deployment.extensions/http-go image updated
```

```
$ kubectl rollout undo deployment http-go  
deployment.extensions/http-go
```

```
$ kubectl exec http-go-7dbcf5877-d6n6p curl 127.0.0.1:8080  
Welcome! http-go:v2
```

```
$ kubectl rollout undo deployment http-go --to-revision=1  
deployment.extensions/http-go
```

애플리케이션 롤링 업데이트와 롤백

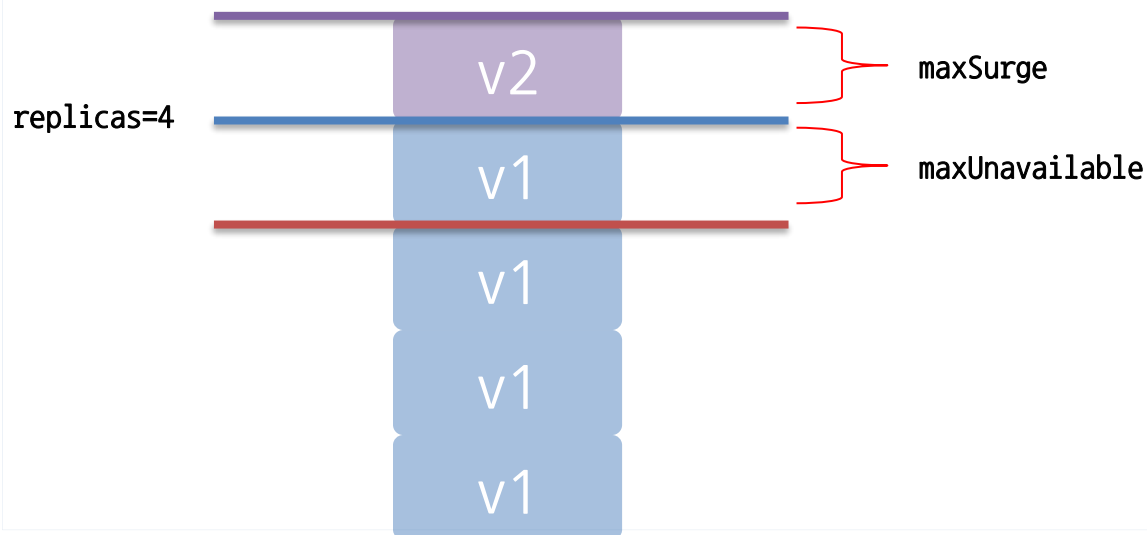
▶ 롤링 업데이터 전략 세부 설정

● maxSurge

- 기본값 25%, 개수로도 설정이 가능
- 최대로 추가 배포를 허용할 개수 설정
- 4개인 경우 25%이면 1개가 설정 (총 개수 5개까지 동시 포드 운영)

● maxUnavailable

- 기본값 25%, 개수로도 설정이 가능
- 동작하지 않는 포드의 개수 설정
- 4개인 경우 25%이면 1개가 설정 (총 개수 4-1개는 운영해야 함)



```
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
```

애플리케이션 롤링 업데이트와 롤백

▶ 롤아웃 일시중지와 재시작

- 업데이트 중에 일시정지하길 원하는 경우
 - \$ kubectl rollout pause deployment http-go
- 업데이트 일시중지 중 취소
 - \$ kubectl rollout undo deployment http-go
- 업데이트 재시작
 - \$ kubectl rollout resume deployment http-go

애플리케이션 롤링 업데이트와 롤백

업데이트를 실패한 경우

● 업데이트를 실패하는 케이스

- 부족한 할당량(Insufficient quota)
- 레디네스 프로브 실패(Readiness probe failures)
- 이미지 가져오기 오류(Image pull errors)
- 권한 부족(Insufficient permissions)
- 제한 범위(Limit ranges)
- 응용 프로그램 런타임 구성 오류(Application runtime misconfiguration)

● 업데이트를 실패하는 경우에는 기본적으로 600초 후에 업데이트를 중지한다.

```
spec:  
  processDeadlineSeconds: 600
```

애플리케이션 롤링 업데이트와 롤백

▶▶ 연습문제

- 다음 alpine 이미지를 사용하여 업데이트와 롤백을 실행하라. 모든 revision 내용은 기록되어야 한다.
 - alpine:3.4 이미지를 사용하여 deployment를 생성하라.
 - ✓ Replicas: 10
 - ✓ maxSurge: 50%
 - ✓ maxUnavailable: 50%
 - alpine:3.5 롤링 업데이트를 수행하라.
 - alpine:3.4로 롤백을 수행하라.



Namespaces

Namespaces

네임스페이스란?

- 리소스를 각각의 분리된 영역으로 나누기 좋은 방법
 - 여러 네임스페이스를 사용하면 복잡한 쿠버네티스 시스템을 더 작은 그룹으로 분할
 - 멀티 테넌트(Multi-tenant) 환경을 분리하여 리소스를 생산, 개발, QA 환경 등으로 사용
 - 리소스 이름은 네임스페이스 내에서만 고유 명칭 사용
-
- 현재 클러스터의 기본 네임스페이스 확인하기

```
$ kubectl get ns
```

NAME	STATUS	AGE
application-system	Active	7h3m
default	Active	7h13m
kube-public	Active	7h13m
kube-system	Active	7h13m

Namespaces

▶▶ 각 네임스페이스 상세 내용 확인

- `kubectl get` 을 옵션없이 사용하면 default 네임스페이스에 질의
- 다른 사용자와 분리된 환경으로 타인의 접근을 제한
- 네임스페이스 별로 리소스 접근 허용과 리소스 양도 제어 가능
- `--namespace`나 `-n`을 사용하여 네임스페이스 별로 확인이 가능

```
$ kubectl get po --namespace kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
event-exporter-v0.2.4-5f7d5d7dd4-szrgm	2/2	Running	0	7h17m
fluentd-gcp-scaler-7b895cbc89-qxd5g	1/1	Running	0	7h16m
fluentd-gcp-v3.2.0-qtwf7	2/2	Running	0	7h16m
...				

Namespaces

▶▶▶ YAML 파일로 네임스페이스 만들기

- test_ns.yaml 파일을 생성하고 create 를 사용하여 생성

```
apiVersion: v1
kind: Namespace
metadata:
  # 네임스페이스 이름
  name: test-ns
```

kubectl 명령어로 yaml 없이 바로 네임스페이스 생성 가능
\$ kubectl create namespace "test-namespace"

```
$ kubectl create -f test_ns.yaml
```

```
namespace/test-ns created
```

```
$ kubectl get ns
```

NAME	STATUS	AGE
application-system	Active	7h44m
default	Active	7h54m
kube-public	Active	7h54m
kube-system	Active	7h54m
test-ns	Active	4m17s

Namespaces

▶▶ 전체 네임스페이스 조회

- 전체 네임스페이스를 대상으로 kubectl을 실행하는 방법
- \$ kubectl get pod --all-namespaces

```
$ kubectl get pod --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
application-system	application-controller-manager-0	1/1	Running	0	7h50m
default	http-go-gpu	1/1	Running	0	6h47m
default	http-go	1/1	Running	0	7h10m
default	http-go-v2	1/1	Running	0	7h11m
kube-system	event-exporter-v0.2.4-5f7d5d7dd4-szrgm	2/2	Running	0	8h
kube-system	fluentd-gcp-scaler-7b895cbc89-qxd5g	1/1	Running	0	7h59m
kube-system	fluentd-gcp-v3.2.0-qtwf7	2/2	Running	0	7h59m
kube-system	fluentd-gcp-v3.2.0-xdtjg	2/2	Running	0	7h59m
kube-system	fluentd-gcp-v3.2.0-xwggw5	2/2	Running	0	7h59m
kube-system	heapster-v1.6.0-beta.1-54cb65c898-xg4p2	3/3	Running	0	7h59m
kube-system	kube-dns-autoscaler-76fcd5f658-d7k1l	1/1	Running	0	7h59m
kube-system	kube-dns-b46cc9485-9w2fj	4/4	Running	0	7h59m
kube-system	kube-dns-b46cc9485-lxwc7	4/4	Running	0	8h
kube-system	kube-proxy-gke-standard-cluster-1-default-pool-b1e2cd6b-3r3q	1/1	Running	0	7h59m
kube-system	kube-proxy-gke-standard-cluster-1-default-pool-b1e2cd6b-cb83	1/1	Running	0	8h
kube-system	kube-proxy-gke-standard-cluster-1-default-pool-b1e2cd6b-z2nd	1/1	Running	0	8h
kube-system	l7-default-backend-6f8697844f-xtrt7	1/1	Running	0	8h
kube-system	metrics-server-v0.3.1-5b4d6d8d98-ggjr9	2/2	Running	0	7h59m
kube-system	prometheus-to-sd-5pfw5	1/1	Running	0	7h59m
kube-system	prometheus-to-sd-g66mn	1/1	Running	0	8h
kube-system	prometheus-to-sd-j64x2	1/1	Running	0	8h

Namespaces

▶▶ 연습문제

- 현재 시스템에는 몇 개의 Namespace가 존재하는가?
- kube-system에는 몇 개의 포드가 존재하는가?
- ns-jenkins 네임스페이스를 생성하고 jenkins 포드를 배치하라.
 - pod image: jenkins
 - pod name: jenkins
- coredns는 어느 네임스페이스에 속해있는가?

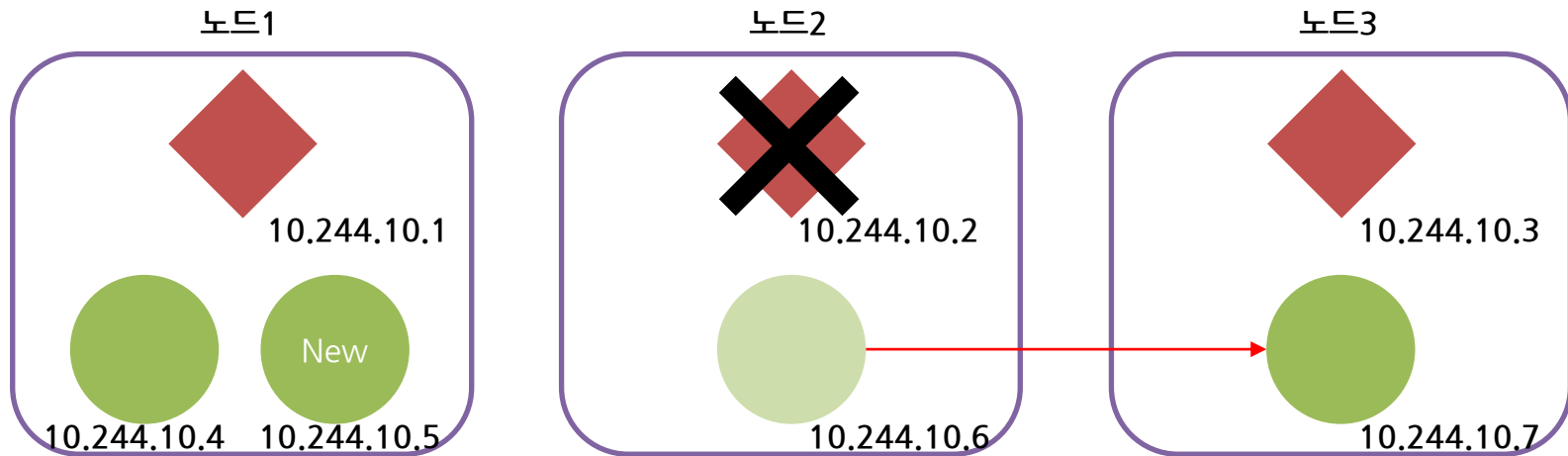


Services

Services

▶ 포드의 문제점

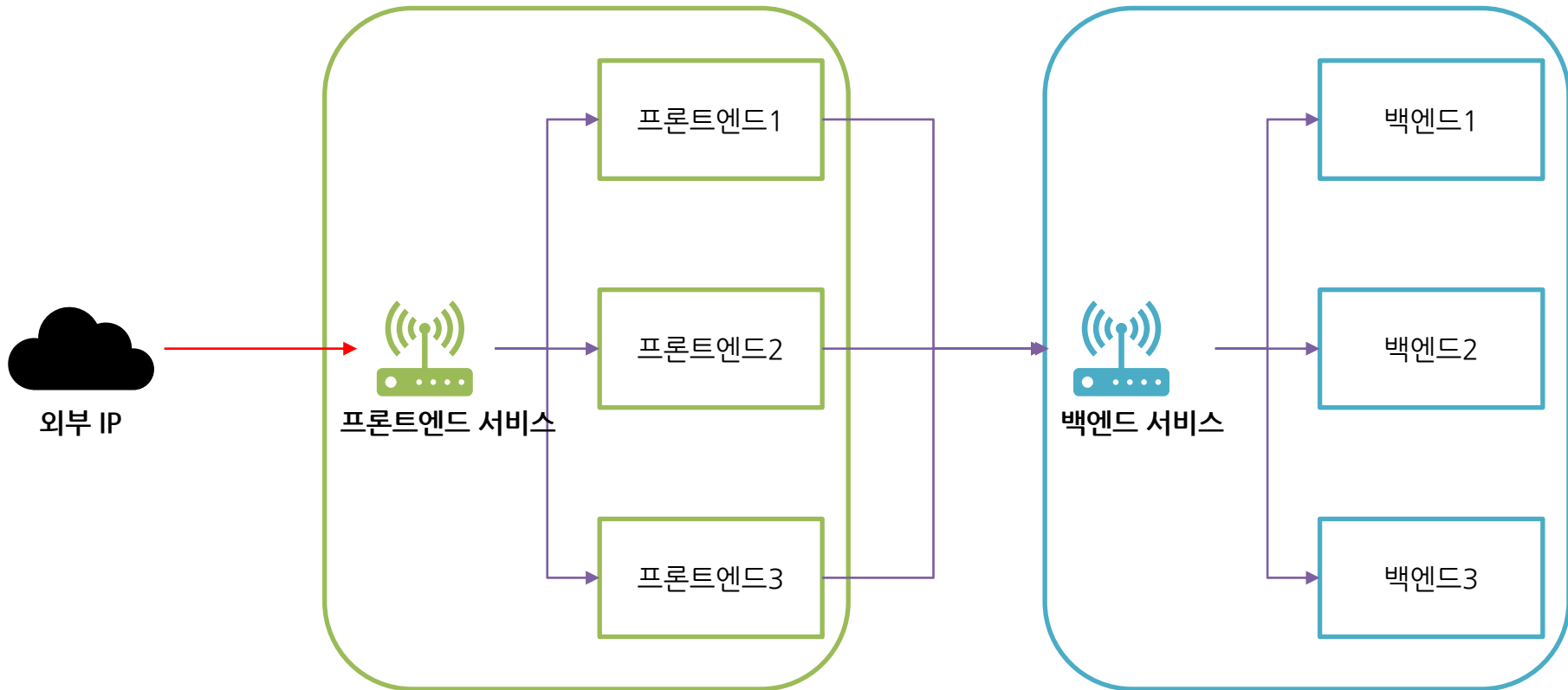
- 포드는 일시적으로 생성한 컨테이너의 집합
- 때문에 포드가 지속적으로 생겨났을 때 서비스를 하기에 적합하지 않음
- IP 주소의 지속적인 변동, 로드밸런싱을 관리해줄 또 다른 개체가 필요
- 이 문제를 해결하기 위해 **서비스**라는 리소스가 존재



Services

서비스의 요구사항

- 외부 클라이언트가 몇 개이든지 프론트엔드 포드로 연결
- 프론트엔드는 다시 백엔드 데이터베이스로 연결
- 포드의 IP가 변경될 때마다 재설정 하지 않도록 해야함



Services

▶ 서비스의 생성방법

- kubectl의 expose가 가장 쉬운 방법
- YAML을 통해 버전 관리 가능

```
$ kubectl create -f http-go-svc.yaml
service/http-go-srv created
```

```
$ kubectl create -f http-go-rs.yaml
replicaset.apps/http-go-rs created
```

```
$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.12.0.1	<none>	443/TCP	53m
http-go-svc	ClusterIP	10.12.1.116	<none>	80/TCP	21m

http-go-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: http-go-svc
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: http-go
```

Services

▶ 서비스의 기능 확인

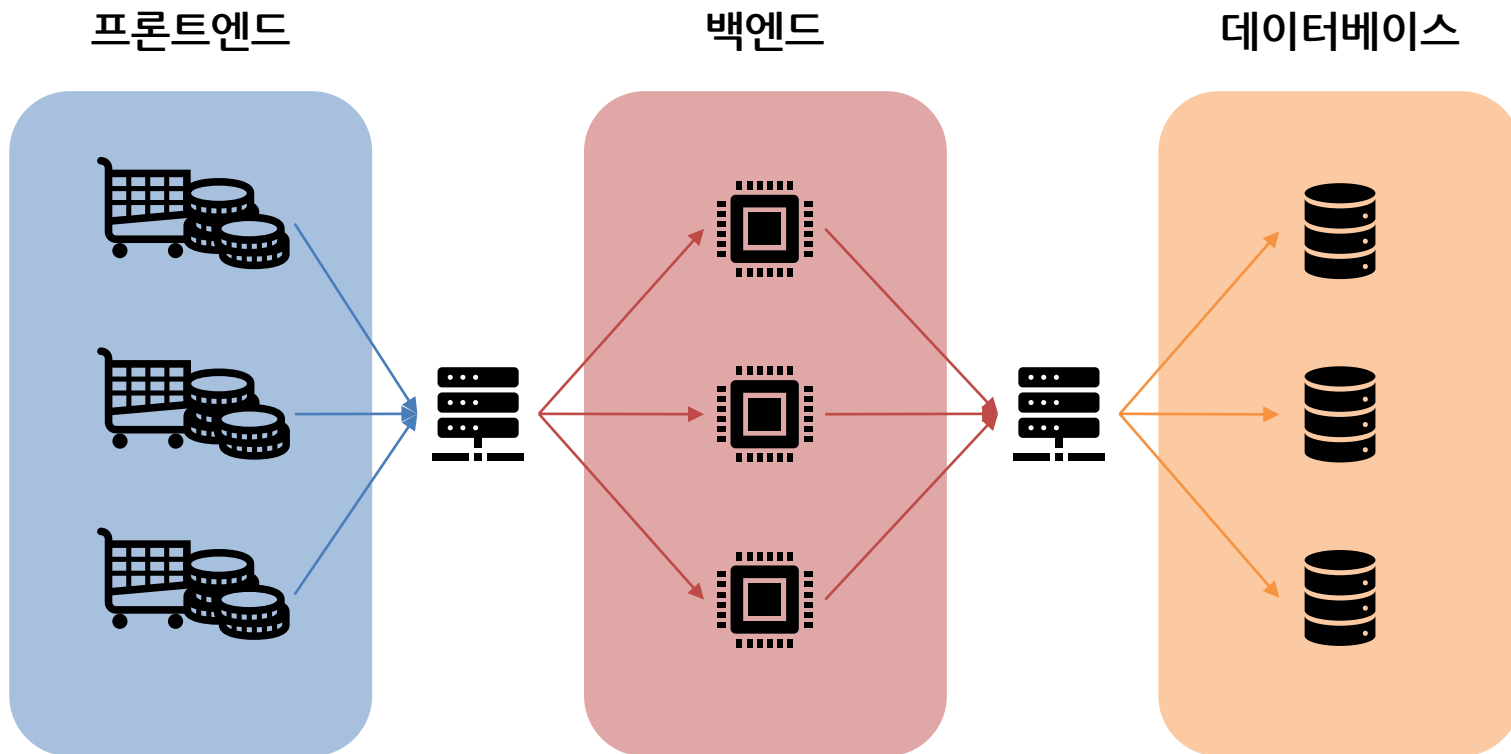
- 서비스를 생성하면 EXTERNAL-IP를 아직 받지 못한 것을 확인
- `kubectl exec <포드 이름> -- curl` 명령어로 확인해보자.

```
$ kubectl exec http-go-rs-4l52m -- curl 10.12.0.237:80 -s  
Welcome! http-go-rs-vsf5n
```


Services

▶ 포드 간의 통신을 위한 ClusterIP

- 다수의 포드를 하나의 서비스로 묶어서 관리



Services

ClusterIP 정의 예

프론트엔드

백엔드

데이터베이스

```
apiVersion: v1
kind: Service
metadata:
  name: back-end
spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80
  selector:
    app: myapp
    type: back-end
```

```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  type: ClusterIP
  ports:
    - targetPort: 3006
      port: 3306
  selector:
    app: mysql
    type: db
```

Services

서비스의 세션 고정하기

- 서비스가 다수의 포드로 구성하면 웹서비스의 세션이 유지되지 않음
- 이를 위해 처음 들어왔던 클라이언트 IP를 그대로 유지해주는 방법이 필요
- `sessionAffinity: ClientIP`라는 옵션을 주면 해결 완료!

```
$ kubectl exec http-go-rs-4l52m -- curl 10.12.0.237:80
```

```
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left  Speed
100  27    0    27    0    0  1492    0 --:--:-- --:--:-- --:--:-- 1588
```

Welcome! http-go-rs-vs5n

```
$ kubectl exec http-go-rs-4l52m -- curl 10.12.0.237:80
```

```
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left  Speed
100  27    0    27    0    0  1492    0 --:--:-- --:--:-- --:--:-- 1588
```

Welcome! http-go-rs-vs5n

```
$ kubectl exec http-go-rs-4l52m -- curl 10.12.0.237:80
```

```
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left  Speed
100  27    0    27    0    0  1492    0 --:--:~ --:~:~ --:~:~ 1588
```

Welcome! http-go-rs-vs5n

모두 같은 pod로만 요청

```
apiVersion: v1
kind: Service
metadata:
  name: http-go-svc
spec:
  sessionAffinity: ClientIP
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: http-go
```

affinity

1. 친밀감
2. (밀접한) 관련성, 친연성

Services

▶ 다중 포트 서비스 방법

- 포트에 그대로 나열해서 사용

```
$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.12.0.1	<none>	443/TCP	81m
http-go-svc	ClusterIP	10.12.4.100	<none>	80/TCP,443/TCP	5m7s

```
apiVersion: v1
kind: Service
metadata:
  name: http-go-svc
spec:
  sessionAffinity: ClientIP
  ports:
    - name: http
      port: 80
      targetPort: 8080
    - name: https
      port: 443
      targetPort: 8443
  selector:
    app: http-go
```

Services

▶ 서비스하는 IP 정보 확인

- 서비스 세부 사항에는 연결될 IP에 대한 정보가 존재

```
$ kubectl describe svc http-go-svc
Name:                http-go-svc
Namespace:           default
Labels:              <none>
Annotations:         <none>
Selector:            app=http-go
Type:                ClusterIP
IP:                 10.12.4.100
Port:                http 80/TCP
TargetPort:          8080/TCP
Endpoints:           10.8.1.2:8080,10.8.1.3:8080,10.8.1.4:8080
Port:                https 443/TCP
TargetPort:          8443/TCP
Endpoints:           10.8.1.2:8443,10.8.1.3:8443,10.8.1.4:8443
Session Affinity:    ClientIP
Events:              <none>
```

Services

외부 IP 연결 설정 YAML

- Service와 Endpoints 리소스 모두 생성 필요

external-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  ports:
    - port: 80
```

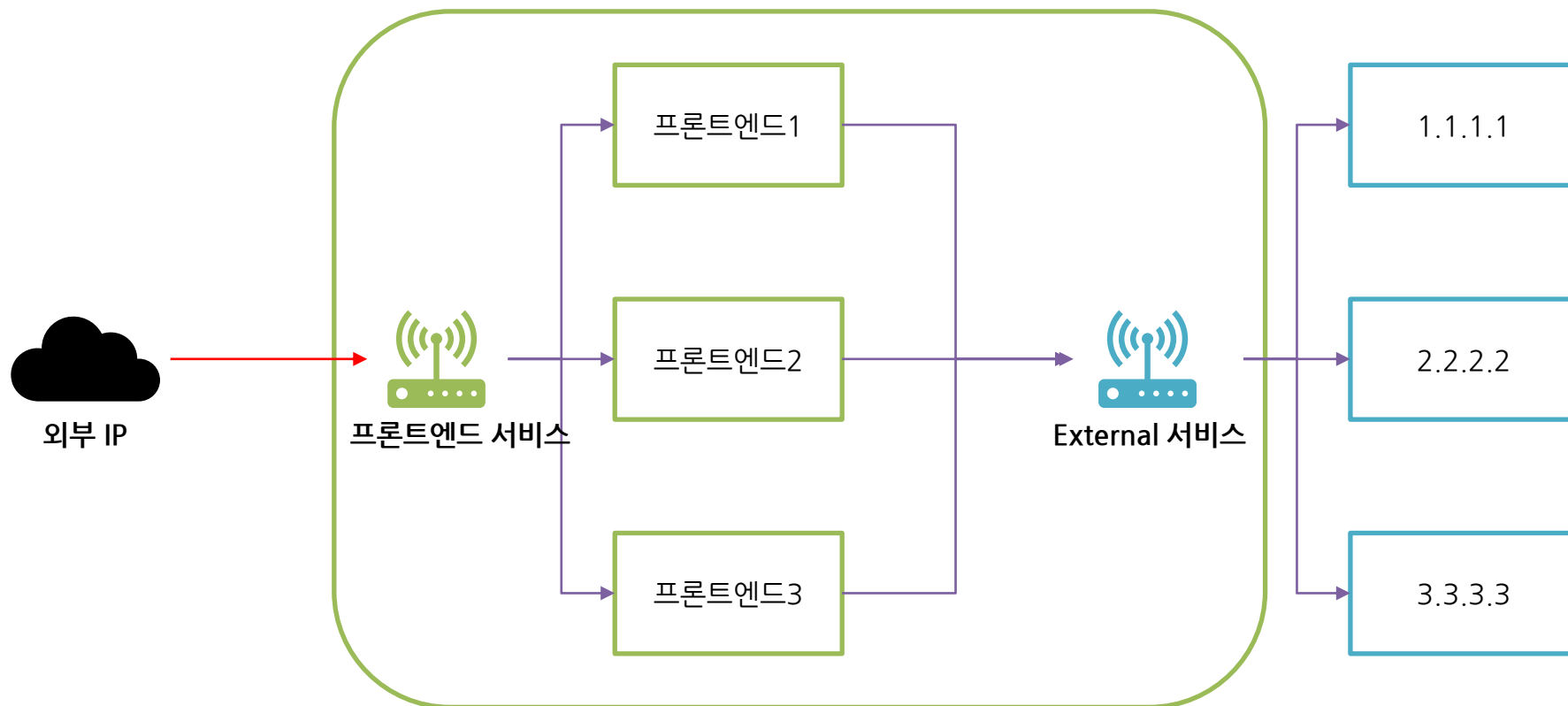
external-endpoint.yaml

```
apiVersion: v1
kind: Endpoints
metadata:
  name: external-service
subsets:
  - addresses:
    - ip: 11.11.11.11
    - ip: 22.22.22.22
  ports:
    - port: 80
```

Services

외부 IP 연결 설정 YAML

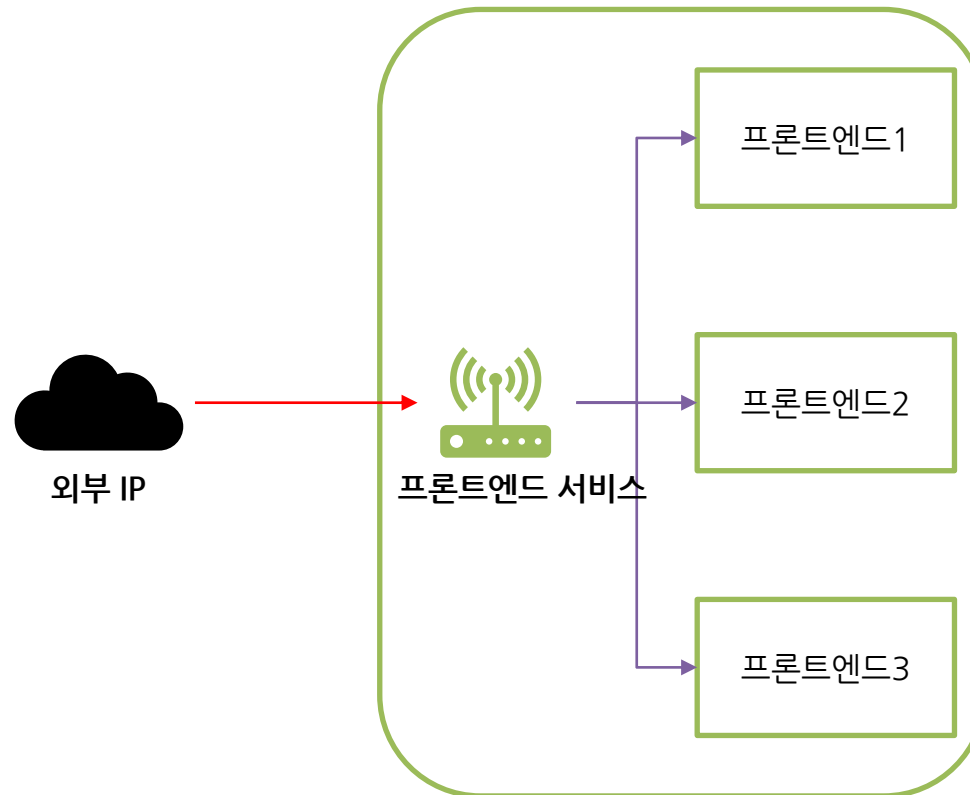
- Service와 Endpoints 연결 구조



Services

서비스 노출하는 세 가지 방법

- NodePort: 노드의 자체 포트를 사용하여 포드로 리다이렉션
- LoadBalancer: 외부 게이트웨이를 사용해 노드 포트로 리다이렉션
- Ingress: 하나의 IP 주소를 통해 여러 서비스를 제공하는 특별한 메커니즘



Services

▶ 노드포트 생성하기

- 서비스 yaml 파일을 작성
- type에 NodePort를 지정
- 30000-32767포트만 사용가능

http-go-np.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: http-go-svc
spec:
  type: NodePort
  ports:
    - port: 80 # 서비스의 포트
      targetPort: 8080 # 포드의 포트
      nodePort: 30001 # 최종적으로 서비스되는 포트
  selector:
    app: http-go
```

\$ kubectl get svc

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.12.0.1	<none>	443/TCP	159m
http-go-np	NodePort	10.12.15.249	<none>	80:30001/TCP	7s
http-go-svc	ClusterIP	10.12.4.100	<none>	80/TCP,443/TCP	83m

Services

▶ 노드포트 생성하기

- GCP에서 방화벽 해제 후 노드로 직접 접속

```
$ gcloud compute firewall-rules create http-go-svc-rule --allow=tcp:30001
Creating firewall...⌘Created [https://www.googleapis.com/compute/v1/projects...].
Creating firewall...done.
```

NAME	NETWORK	DIRECTION	PRIORITY	ALLOW	DENY	DISABLED
http-go-svc-rule	default	INGRESS	1000	tcp:30001		False

```
$ kubectl get node -o wide
```

NAME	STATUS	ROLES	...	EXTERNAL-IP	...
gke-standard-cluster-1-default-pool-d820cb23-lk38	Ready	<none>	...	34.67.21.139	...
gke-standard-cluster-1-default-pool-d820cb23-rhx3	Ready	<none>	...	34.67.112.205	...
gke-standard-cluster-1-default-pool-d820cb23-zcfb	Ready	<none>	...	35.224.244.152	...

```
$ curl 34.67.21.139:30001
Welcome! http-go-rs-5pspr
```

Services

노드포트 서비스의 패킷 흐름

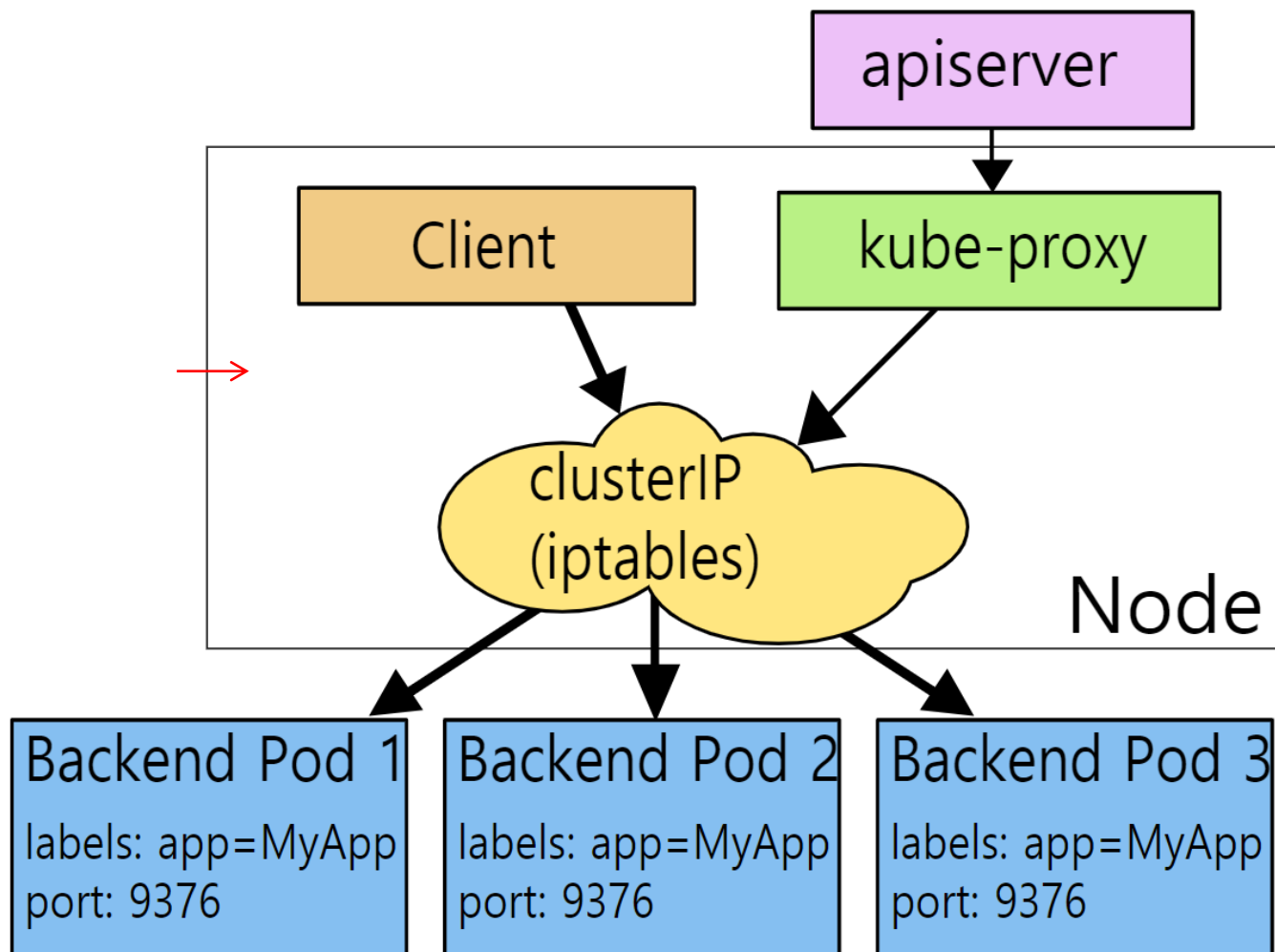


그림 출처: <https://kubernetes.io/docs/concepts/services-networking/service/>

Services

노드포트를 활용한 로드밸런싱

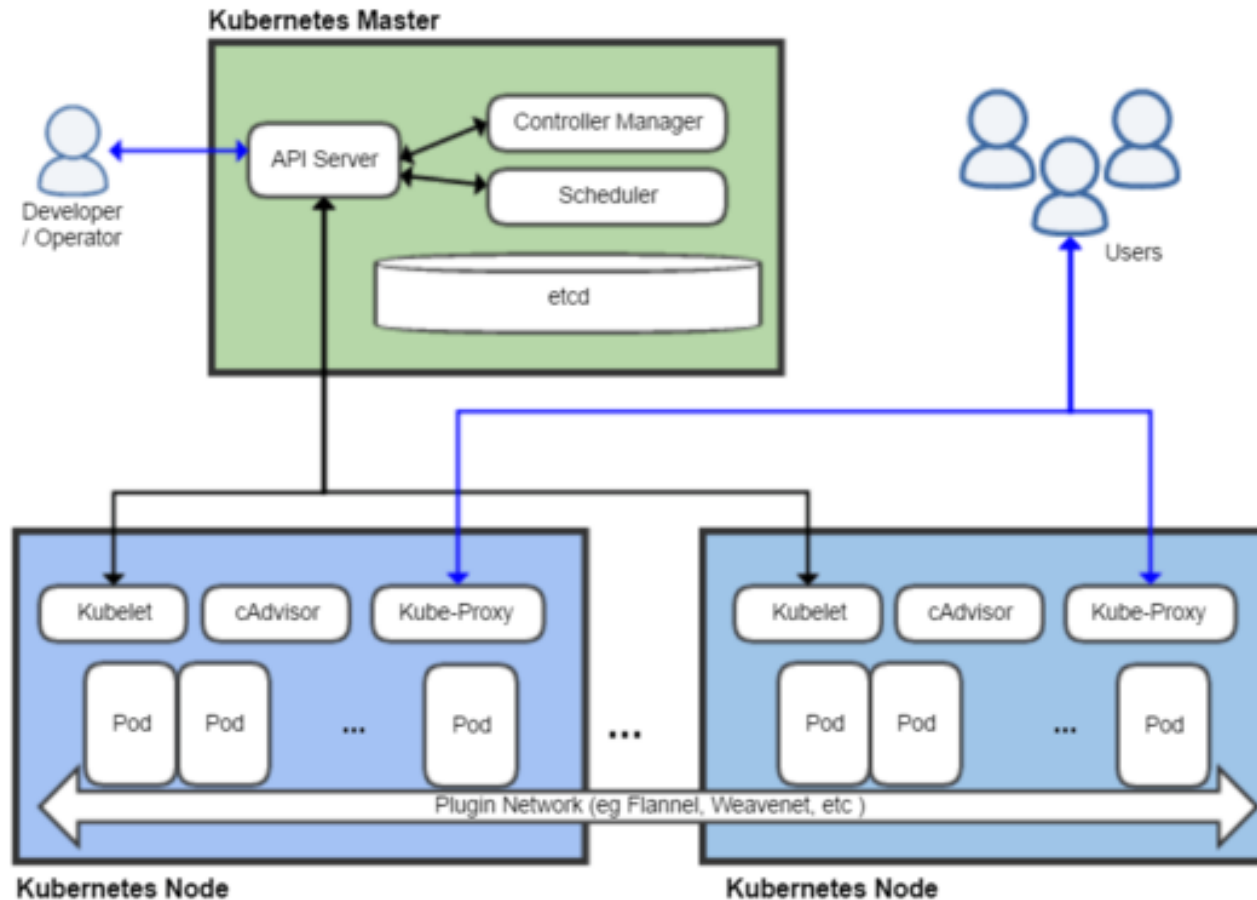


그림 출처: <https://en.wikipedia.org/wiki/Kubernetes>

Services

▶ 로드밸런스 생성하기

- NodePort 서비스의 확장된 서비스
- 클라우드 서비스에서 사용 가능
- yaml 파일에서 타입을 NodePort 대신 LoadBalancer를 설정
- 로드 밸런서의 IP 주소를 통해 서비스에 액세스

http-go-lb.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: http-go-lb
spec:
  type: LoadBalancer
  ports:
    - port: 80 # 서비스의 포트
      targetPort: 8080 # 포드의 포트
  selector:
    app: http-go
```

최종적으로 서비스할 포트

포드의 포트

Services

▶ 로드밸런스 생성하기

- 로드밸런스가 잘 생성됐는지 확인
- 30533포트는 지정해주지 않으면 임의로 생성됨

```
$ kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.12.0.1	<none>	443/TCP	3h19m
http-go-loadbalancer	LoadBalancer	10.12.14.248	34.68.131.99	80:30533/TCP	49s

```
$ curl 34.68.131.99
```

```
Welcome! http-go-rs-4152m
```

Services

로드밸런스 서비스의 패킷 흐름

- 클러스터에 로드밸런싱을 해주는 개체 필요
- 로드밸런스는 노드포트의 기능을 포함

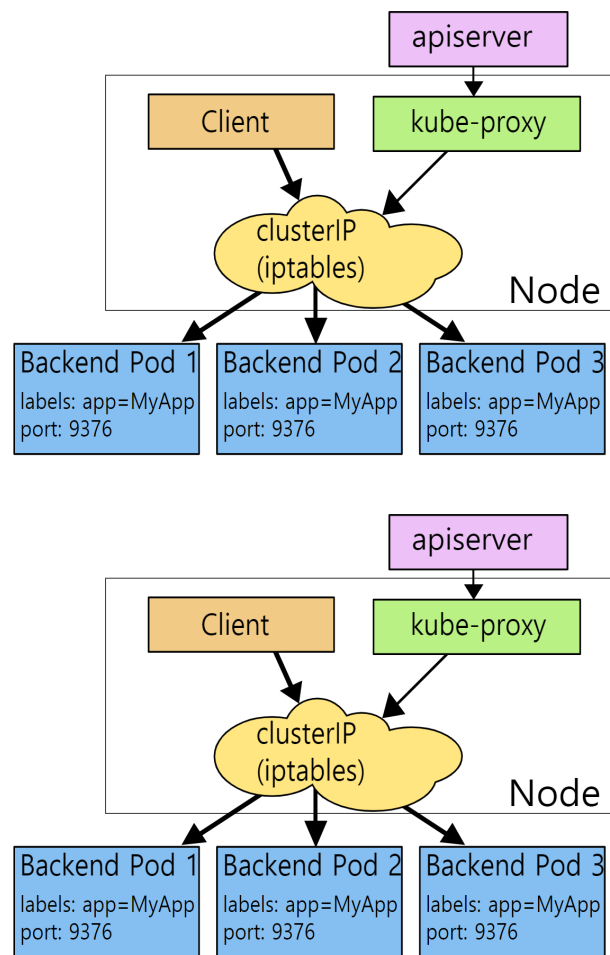


그림 출처: <https://kubernetes.io/docs/concepts/services-networking/service/>

Services

연습문제

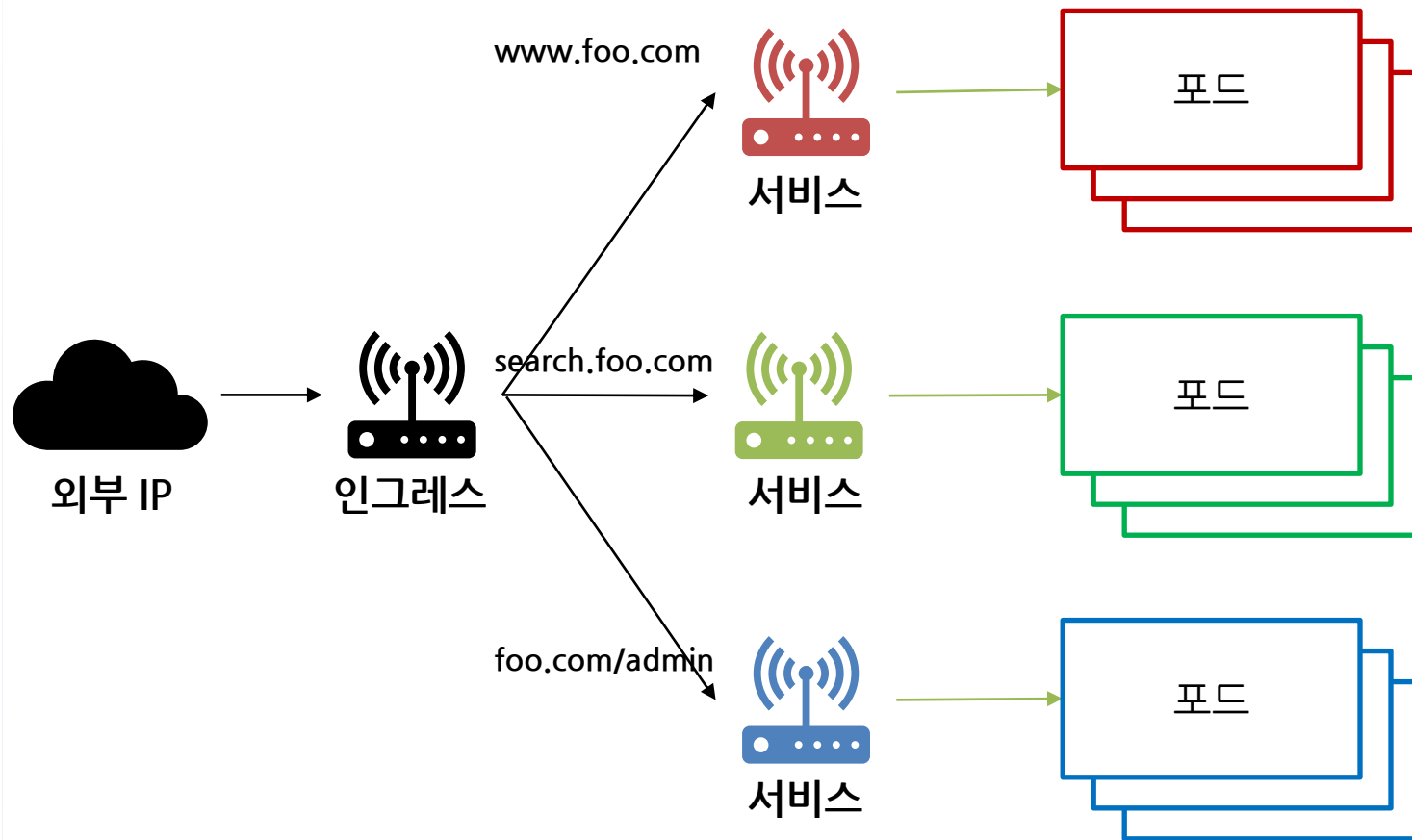
- tomcat을 노드포트로 서비스하기(30002번 포트 사용)
- tomcat을 로드밸런스로 서비스하기(80번 포트 사용)

Services

인그레스의 필요성

- 인그레스는 하나의 IP이나 도메인으로 다수의 서비스 제공

ingress
(어떤 장소에) 들어감, 입장; 들어갈 수 있는 권리, 입장권



Services

▶ 인그레스 생성하기

http-go-ingress.yaml

```
apiVersion: extensions/v1beta1
```

```
kind: Ingress
```

```
metadata:
```

```
  name: ingress
```

```
spec:
```

```
  rules:
```

```
  - host: gasbugs.com
```

도메인 이름 일치해야 함
(그렇지 않은 경우 404페이지 출력)

```
    http:
```

```
      paths:
```

```
      - path: /
```

```
        backend:
```

```
          serviceName: http-go-np
```

```
          servicePort: 8080
```

연결할 서비스 설정

Services

▶ 인그레스 정보 확인 및 접속

- 접속을 위해서는 반드시 룰에 일치되어야 하므로 HTTP 요청의 host가 gasbugs.com 값을 가질 수 있도록 설정 (/etc/hosts를 변경)
- 만약 ADDRESS가 장시간 설정되지 않는다면 연결할 노드포트와 포드가 제대로 올라와있는지 확인해야 함.

```
$ kubectl get ingresses
```

NAME	HOSTS	ADDRESS	PORTS	AGE
http-go	gasbugs.com	34.98.103.218	80	9m49s

```
$ sudo vim /etc/hosts
```

```
[...]
```

```
34.98.103.218 gasbugs.com
```

```
$ curl http://gasbugs.com
```

```
Welcome! http-go-rs-4l52m
```

Services

다수의 서비스를 제공하고 싶을 때는?

```
$ kubectl get ingress
```

NAME	HOSTS	ADDRESS
http-go	www.gasbugs.com,dict.gasbugs.com,map.gasbugs.com	
34.98.103.218	80	80s

```
$ sudo vim /etc/hosts
```

```
[...]  
34.98.103.218 www.gasbugs.com dict.gasbugs.com map.gasbugs.com
```

```
$ curl www.gasbugs.com
```

```
Welcome! http-go-rs-vs5n
```

```
$ curl dict.gasbugs.com
```

```
Welcome! http-go-rs-4152m
```

```
$ curl map.gasbugs.com
```

```
Welcome! http-go-rs-4152m
```

```
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: ingress  
spec:  
  rules:
```

```
- host: www.gasbugs.com  
  http:  
    paths:  
      - path: /  
        backend:  
          serviceName: http-go-np  
          servicePort: 8080  
- host: dict.gasbugs.com  
  http:  
    paths:  
      - path: /  
        backend:  
          serviceName: http-go-np  
          servicePort: 8080  
- host: map.gasbugs.com  
  http:  
    paths:  
      - path: /  
        backend:  
          serviceName: http-go-np  
          servicePort: 8080
```

Services

인그레스 HTTPS 서비스하기

- TLS 인증성 생성

```
$ openssl genrsa -out tls.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++++
.....+++++
e is 65537 (0x010001)

$ openssl req -new -x509 -key tls.key -out tls.cert -days 360 -subj /CN=www.gasbugs.com

$ kubectl create secret tls tls-secret --cert=tls.cert --key=tls.key
secret/tls-secret created

$ curl -k https://www.gasbugs.com
Welcome! http-go-rs-vs5n
```

-k 옵션: Allow connections to SSL sites without certs (H)

Services

▶▶ 연습문제

- tomcat, http-go를 ingress로 서비스하기
(tomcat.example.com, http-go.example.com 도메인 사용)
- GKE에서 인그레스를 활용한 로드밸런싱 프로세스 확인
 - <https://www.notion.so/gasbugs/1ad380a39042434ba71ae286c1560af8>
 - <https://bit.ly/2Ju0k4Y> (단축url)



Network

Network

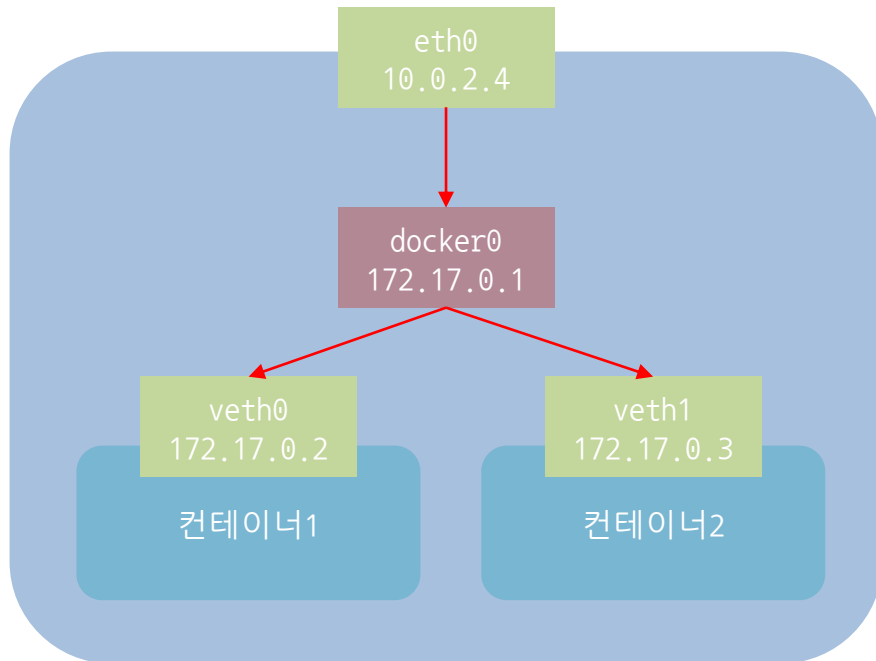
▶ 쿠버네티스 네트워크 모델

- 한 포드에 있는 다수의 컨테이너끼리 통신
 - 포드끼리 통신
 - 포드와 서비스 사이의 통신
 - 외부 클라이언트와 서비스 사이의 통신
-
- 실습 전 설치:
 - `sudo apt install net-tools`

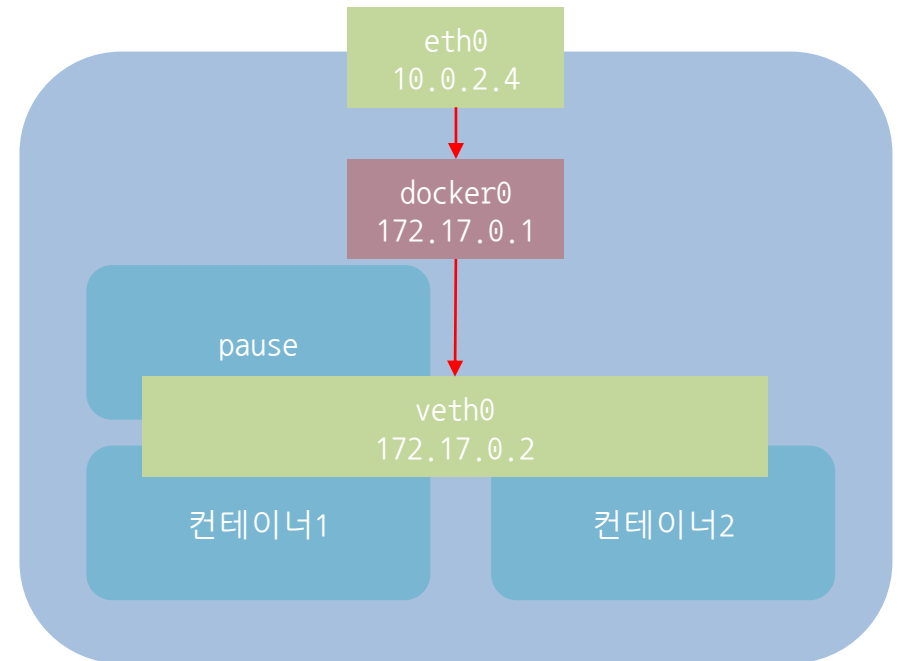
Network

한 포드에 있는 다수의 컨테이너끼리 통신

- pause 명령을 실행해 아무 동작을 하지 않는 빈 컨테이너를 생성
- 인터페이스를 공유
- 포트를 겹치게 구성하지 못하는 것이 특징



도커의 네트워크 구조



컨테이너 간의 인터페이스를 공유하는 구조

Network

▶ 한 포드에 있는 다수의 컨테이너끼리 통신

- Docker의 기능을 사용해 쿠버네티스 컨테이너를 관찰
- 각 포드마다 하나의 pause 이미지 실행

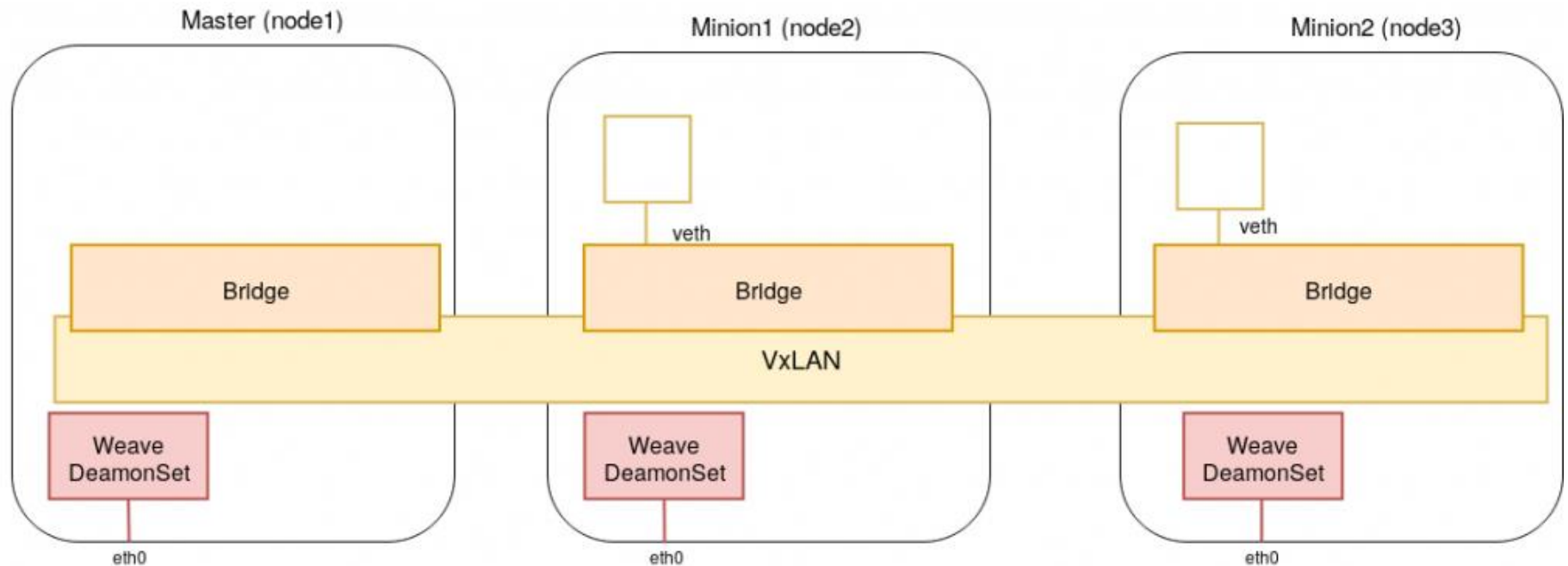
```
$ sudo docker ps | grep pause
```

f349c5fb5fb2	k8s.gcr.io/pause:3.1	"/pause"	4 days ago	Up 4 days
k8s_POD_coredns-6955765f44-qvw44_kube-system_28d0bac3-adb6-4e53-ba9f-7e13b40887a6_0				
cae38b08bed7	k8s.gcr.io/pause:3.1	"/pause"	7 days ago	Up 7 days
k8s_POD_weave-net-7c9fc_kube-system_c7dd9116-cf33-4062-af4f-f9c954feda46_0				
5ddad448f55c	k8s.gcr.io/pause:3.1	"/pause"	7 days ago	Up 7 days
k8s_POD_kube-proxy-d8cnw_kube-system_4fe5f7ee-4757-4cb3-80e5-583620bafe96_1				
04c042f26873	k8s.gcr.io/pause:3.1	"/pause"	7 days ago	Up 7 days
k8s_POD_kube-controller-manager-master_kube-system_27d0c51886b29063a1ed2ae1755a84f9_1				
3c12cc55ced6	k8s.gcr.io/pause:3.1	"/pause"	7 days ago	Up 7 days
k8s_POD_kube-scheduler-master_kube-system_e3025acd90e7465e66fa19c71b916366_1				
d60b1fd5a87e	k8s.gcr.io/pause:3.1	"/pause"	7 days ago	Up 7 days
k8s_POD_kube-apiserver-master_kube-system_1a256086d907b42ba13bb9bb0017fea8_1				
a7cf5791f354	k8s.gcr.io/pause:3.1	"/pause"	7 days ago	Up 7 days
k8s_POD_etcd-master_kube-system_ae5d8b8743f7d6399fcdd2fc8bf7dad5_1				

Network

▶ 포드끼리 통신

- 포드끼리의 통신을 위해서는 CNI 플러그인이 필요
- ACI, AOS, AWS VPN CNI, CNI-Genie, GCE, flannel, Weave Net, Calico 등



Weavenet 네트워크 모델

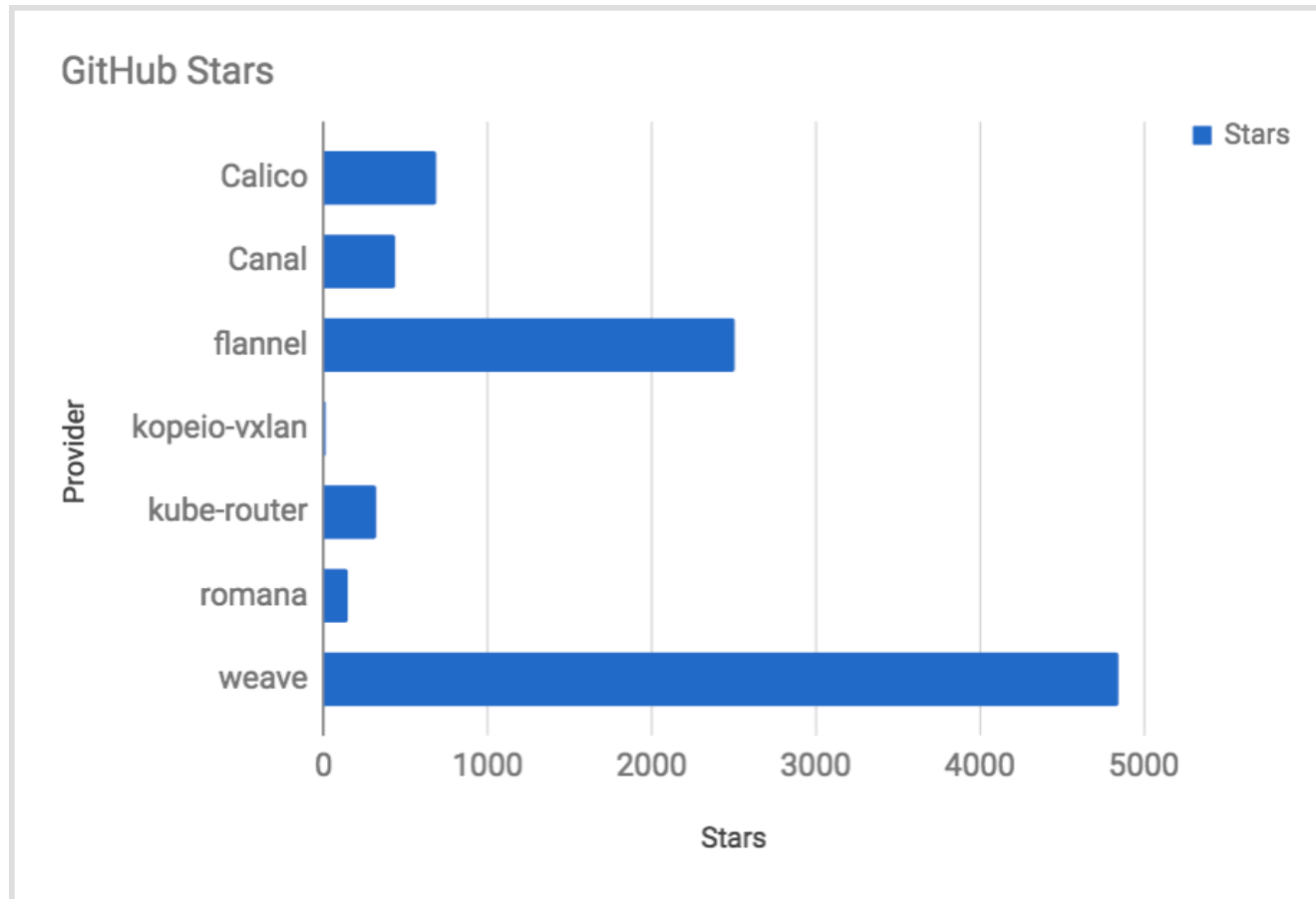
그림출처: <https://www.objectif-libre.com/en/blog/2018/07/05/k8s-network-solutions-comparison/>

Network

▶ 포드끼리 통신

- CNI 플러그인 통계

- <https://chrislovecnm.com/kubernetes/cni/choosing-a-cni-provider/>

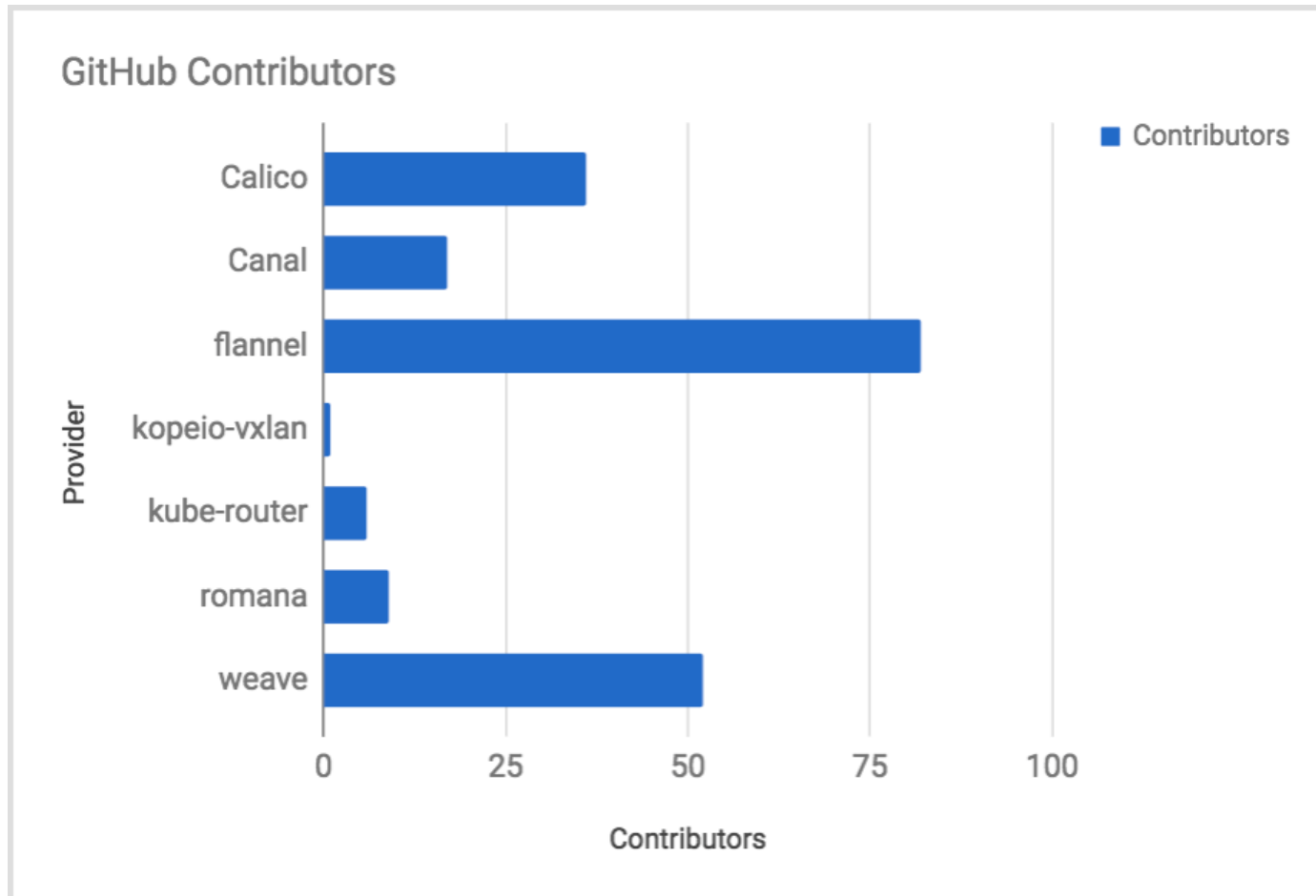


Network

포드끼리 통신

- CNI 플러그인 통계

- <https://chrislovecnm.com/kubernetes/cni/choosing-a-cni-provider/>



Network

포드끼리 통신

- CNI 플러그인 지원 목록

➤ <https://chrislovecnm.com/kubernetes/cni/choosing-a-cni-provider/>

Provider	Network Model	Route Distribution	Network Policies	Mesh	External Datastore	Encryption	Ingress/Egress Policies	Commercial Support
Calico	Layer 3	Yes	Yes	Yes	Etcd ¹	Yes	Yes	Yes
Canal	Layer 2 vxlan	N/A	Yes	No	Etcd ¹	No	Yes	No
flannel	vxlan	No	No	No	None	No	No	No
kopeio-networking	Layer 2 vxlan ²	N/A	No	No	None	Yes ³	No	No
kube-router	Layer 3	BGP	Yes	No	No	No	No	No
romana	Layer 3	OSPF	Yes	No	Etcd	No	Yes	Yes
Weave Net	Layer 2 vxlan ⁴	N/A	Yes	Yes	No	Yes	Yes ⁵	Yes

Network

▶ 포드끼리 통신

● CNI 플러그인 지원 기능 설명

기능	설명
Network Model	VXLAN: 캡슐화된 네트워킹으로 이론적으로 속도가 느림 레이어2 네트워킹: 캡슐화되지 않았기 때문에 오버 헤드의 영향이 없음
Route Distribution	BGP 프로토콜 사용 네트워크 세그먼트에 분할 된 클러스터를 구축하려는 경우 사용 인터넷에서 라우팅 및 연결 가능성 정보를 교환하도록 설계된 외부 게이트웨이 프로토콜
Network Policies	네트워크 정책을 사용하여 포드가 서로 통신 할 수있는 규칙을 적용
Mesh Networking	쿠버네티스 클러스터 간 " Pod to Pod " 네트워킹이 가능 Kubernetes 페레이션이 아니라 포드 간의 순수한 네트워킹입니다.
Encyption	네트워크 컨트롤 플레인을 암호화하여 모든 TCP 및 UDP 트래픽을 암호화
Ingress / Egress Policies	들어오거나 나가는 패킷에 대한 통신 제어

Network

▶ 포드끼리 통신

● 프로세스 및 포트 확인

```
$ sudo netstat -antp | grep weaver
```

tcp	0	0	127.0.0.1:6784	0.0.0.0:*	LISTEN	19979/weaver
tcp	0	0	10.0.2.15:39974	10.96.0.1:443	ESTABLISHED	19819/weave-npc
tcp	0	0	10.0.2.15:39982	10.96.0.1:443	ESTABLISHED	19979/weaver
tcp6	0	0	:::6781	:::*	LISTEN	19819/weave-npc
tcp6	0	0	:::6782	:::*	LISTEN	19979/weaver
tcp6	0	0	:::6783	:::*	LISTEN	19979/weaver
tcp6	0	0	10.0.2.15:6783	10.0.2.4:58047	ESTABLISHED	19979/weaver
tcp6	0	0	10.0.2.15:6783	10.0.2.5:51067	ESTABLISHED	19979/weaver

```
$ ps -eaf | grep 19979
```

```
root      19979 19754  0   3월11 ?        00:00:16 /home/weave/weaver --port=6783 --  
datapath=datapath --name=8a:fc:37:b7:0b:66 --host-root=/host --http-addr=127.0.0.1:6784 --  
metrics-addr=0.0.0.0:6782 --docker-api= --no-dns --db-prefix=/weavedb/weave-net --ipalloc-  
range=10.32.0.0/12 --nickname=master --ipalloc-init consensus=2 --conn-limit=200 --expect-npc  
10.0.2.4 10.0.2.5  
server1   29246 10259  0 14:45 pts/0    00:00:00 grep --color=auto 19979
```


Network

▶ 포드와 서비스 사이의 통신

- ClusterIP를 생성하면 iptables의 설정 적용

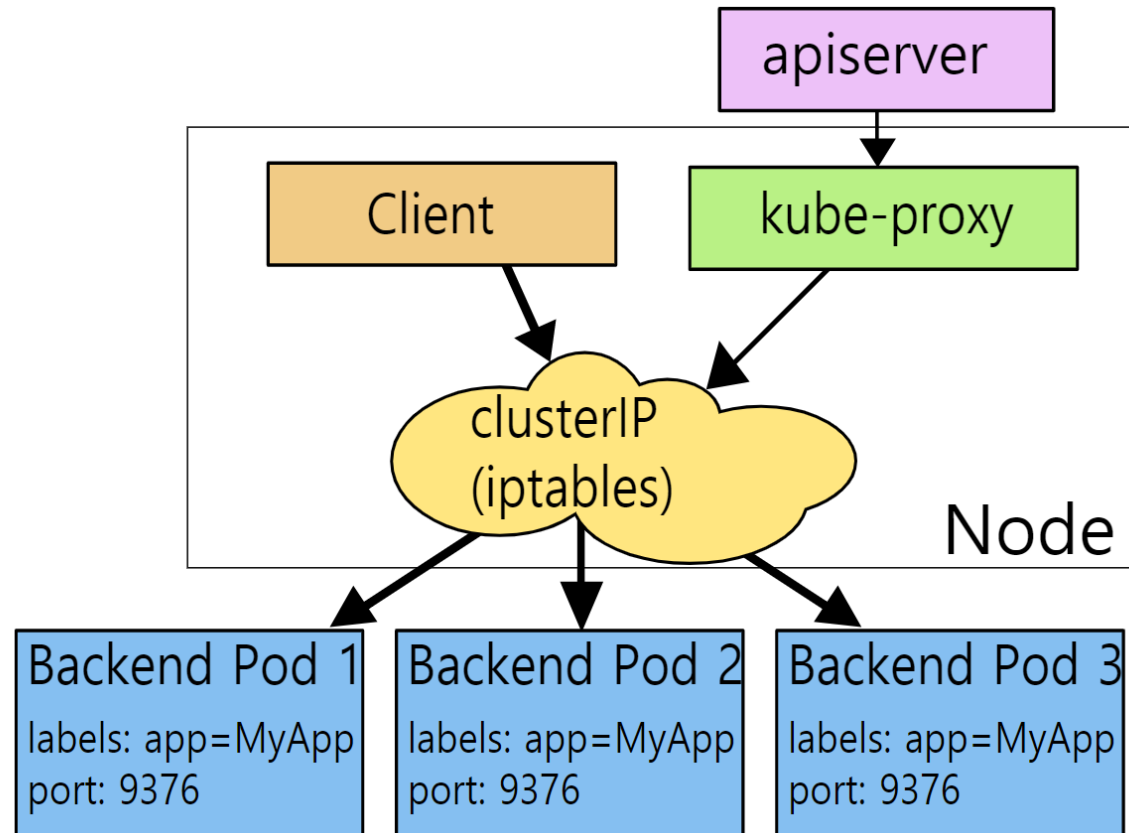


그림 출처: <https://kubernetes.io/docs/concepts/services-networking/service/>

▶ 포드와 서비스 사이의 통신

- Other NF parts
- Other Networking
- basic set of filtering opportunities at the
- network level
- bridge level



Network

▶ 포드와 서비스 사이의 통신

- 다음 그림은 서비스 IP를 통해 10.3.241.152을 요청하는 흐름을 나타냄

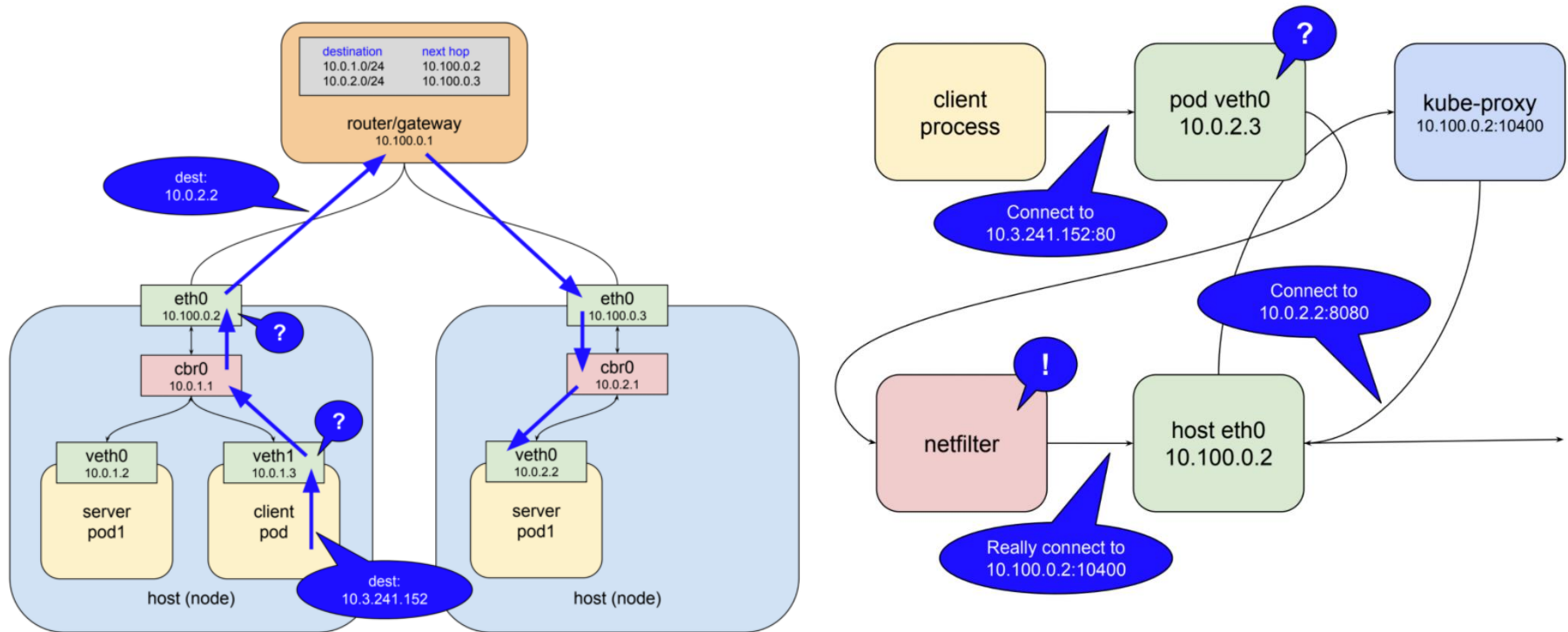


그림 출처: <https://medium.com/google-cloud/understanding-kubernetes-networking-services-f0cb48e4cc82>

Network

▶ 포드와 서비스 사이의 통신

- ClusterIP를 생성하면 iptables의 설정 적용이 됨
- Iptable에서 목록을 확인하는 실습

```
$ kubectl get svc --all-namespaces
```

default	kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP
23h					
kube-system	kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP,53/TCP,9153/TCP
8d					

```
$ sudo iptables -S -t nat | grep 10.96
```

```
-A KUBE-SERVICES -d 10.96.0.1/32 -p tcp -m comment --comment "default/kubernetes:https cluster IP" -m tcp --dport 443 -j KUBE-SVC-NPX46M4PTMTKRN6Y
-A KUBE-SERVICES -d 10.96.0.10/32 -p udp -m comment --comment "kube-system/kube-dns:dns cluster IP" -m udp --dport 53 -j KUBE-SVC-TCOU7JCQXEZGVUNU
-A KUBE-SERVICES -d 10.96.0.10/32 -p tcp -m comment --comment "kube-system/kube-dns:dns-tcp cluster IP" -m tcp --dport 53 -j KUBE-SVC-ERIFXISQEP7F70F4
-A KUBE-SERVICES -d 10.96.0.10/32 -p tcp -m comment --comment "kube-system/kube-dns:metrics cluster IP" -m tcp --dport 9153 -j KUBE-SVC-JD5MR3NA4I4DYORP
```

Network

외부 클라이언트와 서비스 사이의 통신

- 앞서 학습한대로 netfilter와 kube-proxy 기능을 사용해 원하는 서비스 및 포드로 연결

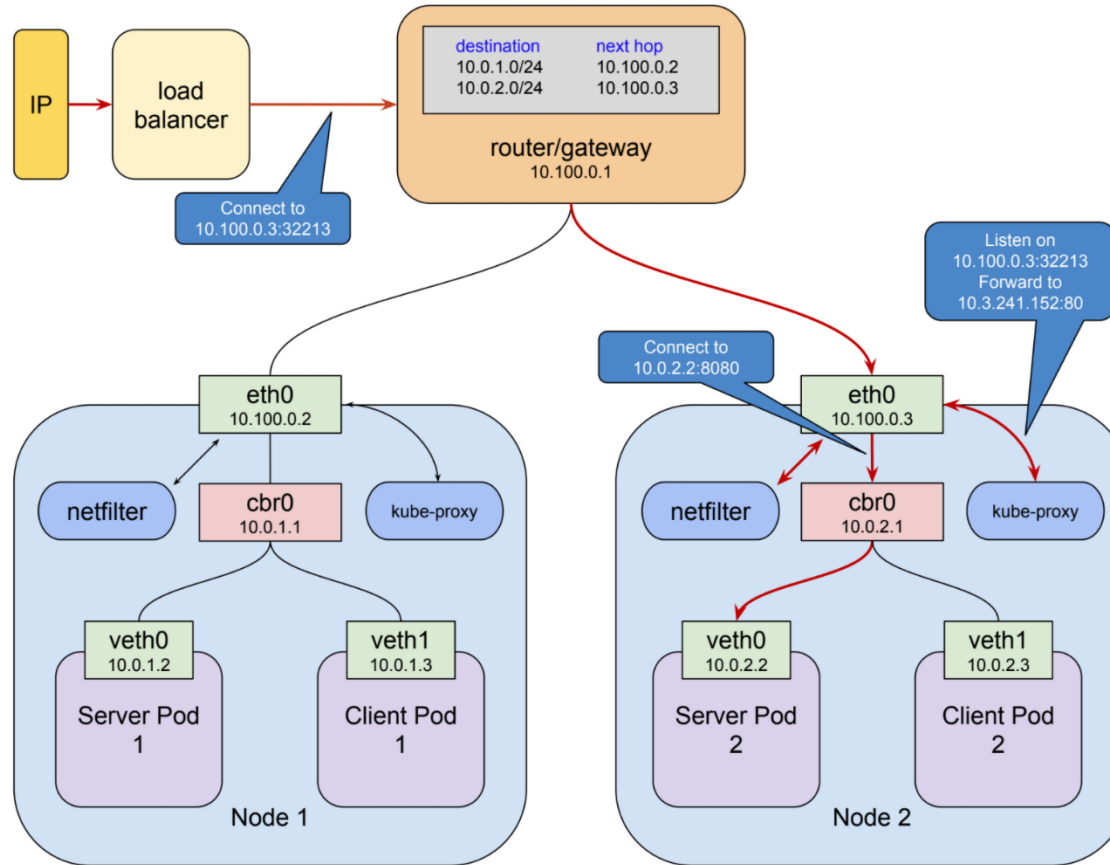


그림 출처: <https://medium.com/google-cloud/understanding-kubernetes-networking-ingress-1bc341c84078>

Network

▶ DNS 서비스를 이용한 서비스 검색

- 서비스를 생성하면, 대응되는 DNS 엔트리가 생성
- 이 엔트리는 <서비스-이름>.<네임스페이스-이름>.svc.cluster.local의 형식을 가짐

```
$ kubectl exec -it http-go-rs-4l52m bash
```

```
root@http-go-rs-4l52m:/# curl http-go-svc.default.svc.cluster.local
Welcome! http-go-rs-5pspr
```

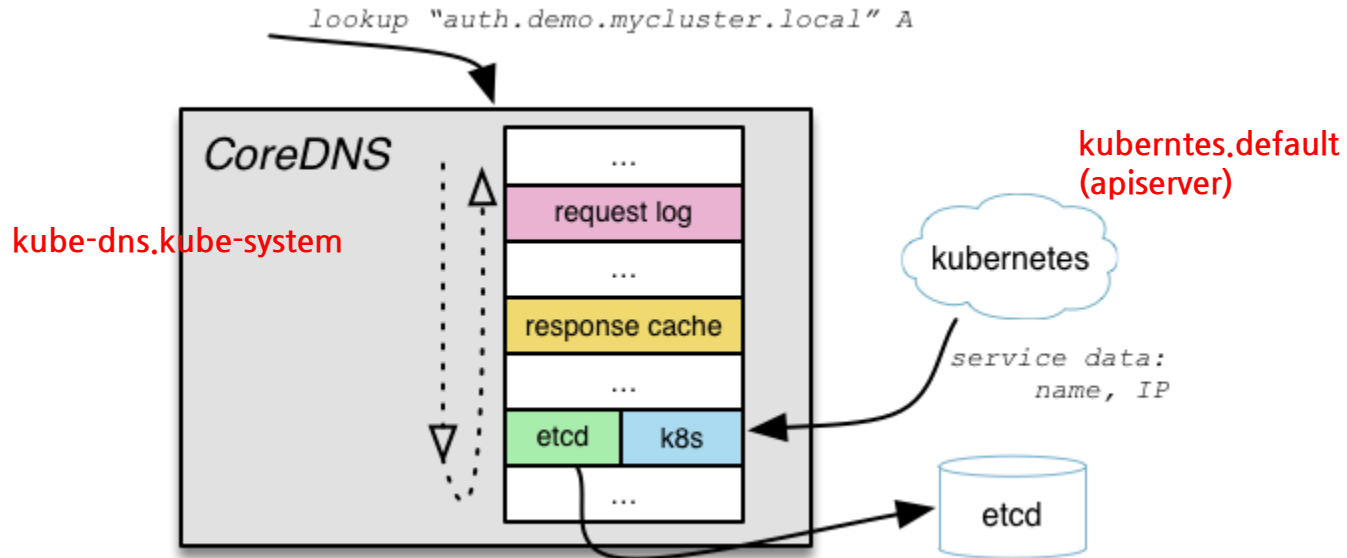
```
root@http-go-rs-4l52m:/# curl http-go-svc.default
Welcome! http-go-rs-5pspr
```

```
root@http-go-rs-4l52m:/# curl http-go-svc
```

Network

CoreDNS

- 내부에서 DNS 서버 역할을 하는 POD이 존재
- 각 미들웨어를 통해 로깅, 캐시, 쿠버네티스를 질의하는 등의 기능을 가짐



미들웨어가 각 기능을 담당하는 CoreDNS 아키텍처

그림 출처: <https://weekly-geekly.github.io/articles/331872/index.html>

Network

CoreDNS

- 해당 DNS에는 configmap 저장소를 사용해 설정 파일을 컨트롤
- Corefile을 통해 현재 클러스터의 NS를 지정

```
$ kubectl get configmap coredns -n kube-system -o yaml
```

```
...
```

```
Corefile: |
  .:53 {
    errors
    health {
      lameduck 5s
    }
    ready
    kubernetes cluster.local in-addr.arpa ip6.arpa {
      pods insecure
      fallthrough in-addr.arpa ip6.arpa
      ttl 30
    }
    prometheus :9153
    forward . /etc/resolv.conf
    cache 30
    loop
    reload
    loadbalance
  }
```

```
...
```


Network

POD에서도 Subdomain을 사용하면 DNS 서비스를 사용 가능

- yaml 파일의 호스트 이름은 포드의 metadata.name을 따름
- 필요한 경우 hostname을 따로 선택 가능
- subdomain은 서브도메인을 지정하는 데 사용
- 서브 도메인을 설정하면 FQDN 사용 가능
- FQDN 형식과 예

`<hostname>.<subdomain>.<namespace>.svc.cluster-domain.example`

↓ ↓ ↓

`busybox-1.default-subdomain.default.svc.cluster-domain.example`

Network

POD에서도 Subdomain을 사용하면 DNS 서비스를 사용 가능

subdomain-svr-pod.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: default-subdomain
spec:
  selector:
    name: busybox
  clusterIP: None
  ports:
    - name: foo # Actually, no port is needed.
      port: 1234
      targetPort: 1234
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox1
  labels:
    name: busybox
spec:
  hostname: busybox-1
  subdomain: default-subdomain
  containers:
    - image: busybox:1.28
      command:
        - sleep
        - "3600"
      name: busybox
---
```

반드시 일치 필요

필요한 경우 다른
hostname 선택 가능

Network

▶▶ 연습문제

- 네임스페이스 blue에 jenkins 이미지를 사용하는 pod-jenkins 디플로이먼트를 생성하고 이를 위한 서비스 srv-jenkins를 생성하라.
- default 네임스페이스의 http-go 이미지의 curl을 사용하여 pod-jenkins:8080을 요청하라.
- `kubectl exec http-go-77cb5c879-29kld -- curl srv-jenkins.blue:8080`



Storage

Storage

볼륨(Volume)

- 컨테이너가 외부 스토리지에 액세스하고 공유하는 방법
 - 포드의 각 컨테이너에는 고유의 분리된 파일 시스템 존재
 - 볼륨은 포드의 컴포넌트이며 포드의 스펙에 의해 정의
 - 독립적인 쿠버네티스 오브젝트가 아니며 스스로 생성, 삭제 불가
 - 각 컨테이너의 파일 시스템의 볼륨을 마운트하여 생성
-
- 볼륨의 종류

임시 볼륨	로컬 볼륨	네트워크 볼륨	네트워크 볼륨 (클라우드 종속적)
emptyDir	hostpath local	iSCSI NFS cephFS glusterFS ...	gcePersistentDisk awsEBS azureFile ...

Storage

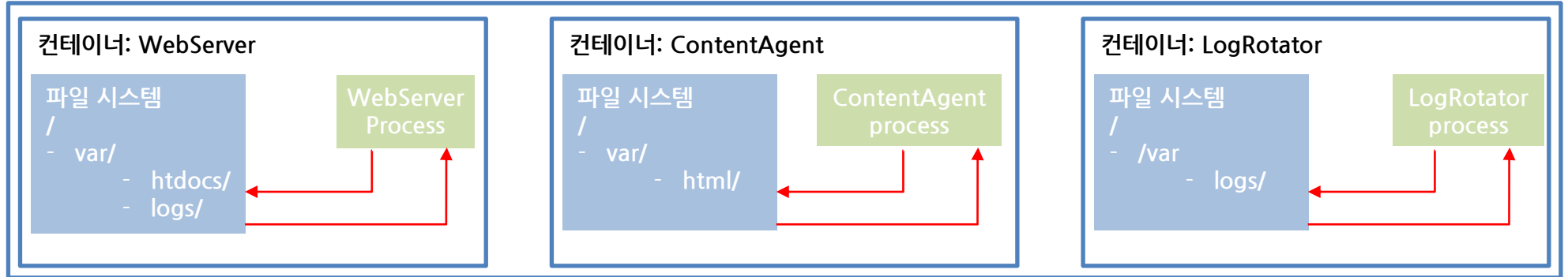
▶ 주요 사용 가능한 볼륨의 유형

- **emptyDir**: 일시적인 데이터 저장, 비어 있는 디렉터리
- **hostPath**: 포드에 호스트 노드의 파일 시스템에서 파일이나 디렉토리를 마운트
- **nfs**: 기존 NFS (네트워크 파일 시스템) 공유가 포드에 장착
- **gcePersistentDisk**: 구글 컴퓨트 엔진 (GCE) 영구디스크 마운트
(awsElasticBlockStore, azureDisk 또한 클라우드에서 사용하는 형태)
- **persistentVolumeClaim**: 사용자가 특정 클라우드 환경의 세부 사항을 모른 채 GCE PersistentDisk 또는 iSCSI 볼륨과 같은 내구성 스토리지를 요구(Claim)할 수 있는 방법
- **configMap, Secret, downwardAPI**: 특수한 유형의 볼륨(나중에 배움)
- 볼륨 관련 레퍼런스
 - <https://kubernetes.io/docs/concepts/storage/volumes/#persistentvolumeclaim>

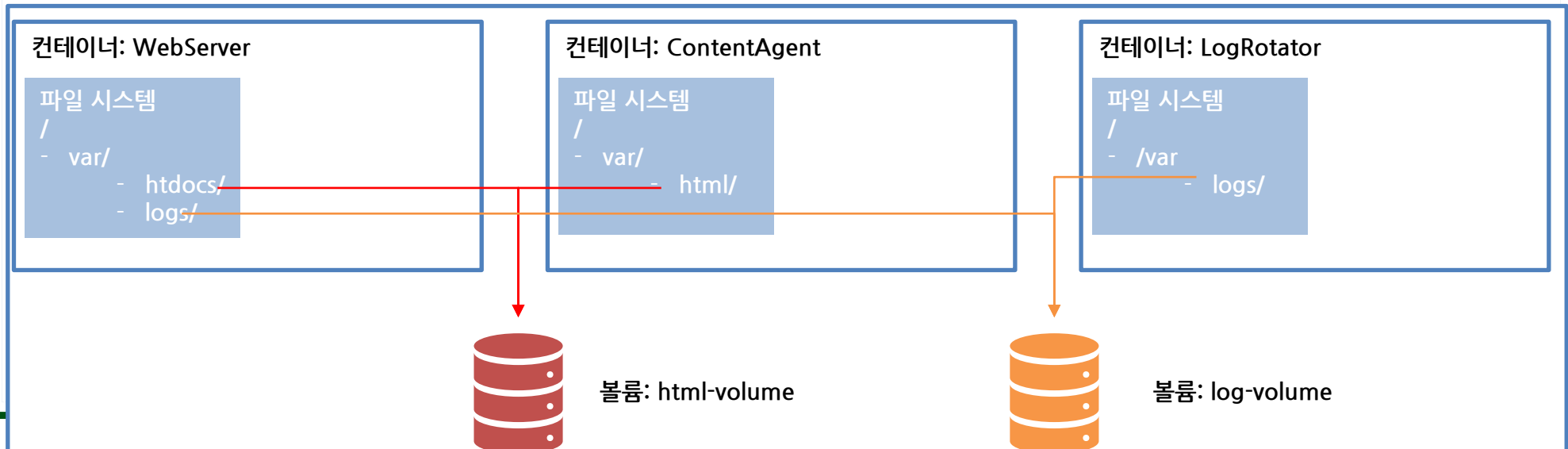
Storage

emptyDir을 활용한 파일 시스템 공유

- 공유 스토리지가 없는 동일한 포드의 세 개의 컨테이너



- 두 개의 볼륨을 공유하는 세 개의 컨테이너



Storage

▶ emptyDir 볼륨 사용하기

- 볼륨을 공유하는 애플리케이션 생성
- docker build -t gasbugs/count .
- docker push gasbugs/count

```
FROM busybox:latest dockerfile

ADD count.sh /bin/count.sh

ENTRYPOINT /bin/count.sh
```

```
#!/bin/bash count.sh
trap "exit" SIGINT
mkdir /var/htdocs

SET=$(seq 0 99999)

for i in $SET

do
    echo "Running loop seq "$i > /var/htdocs/index.html
    sleep 10
done
```


Storage

emptyDir 볼륨 사용하기

```
$ kubectl create -f count-pod.yaml
pod/count created

$ kubectl port-forward count 8080:80 &
Forwarding from 127.0.0.1:8080 -> 80

$ curl 127.0.0.1:8080
Handling connection for 8080
Running loop seq 3
```

각 포드에 마운트 설정

볼륨 설정

count-httpd.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: count
spec:
  containers:
    - image: gasbugs/count
      name: html-generator
      volumeMounts:
        - name: html
          mountPath: /var/htdocs
    - image: httpd
      name: web-server
      volumeMounts:
        - name: html
          mountPath: /usr/local/apache2/htdocs
          readOnly: true
      ports:
        - containerPort: 80
          protocol: TCP
  volumes:
    - name: html
```

Storage

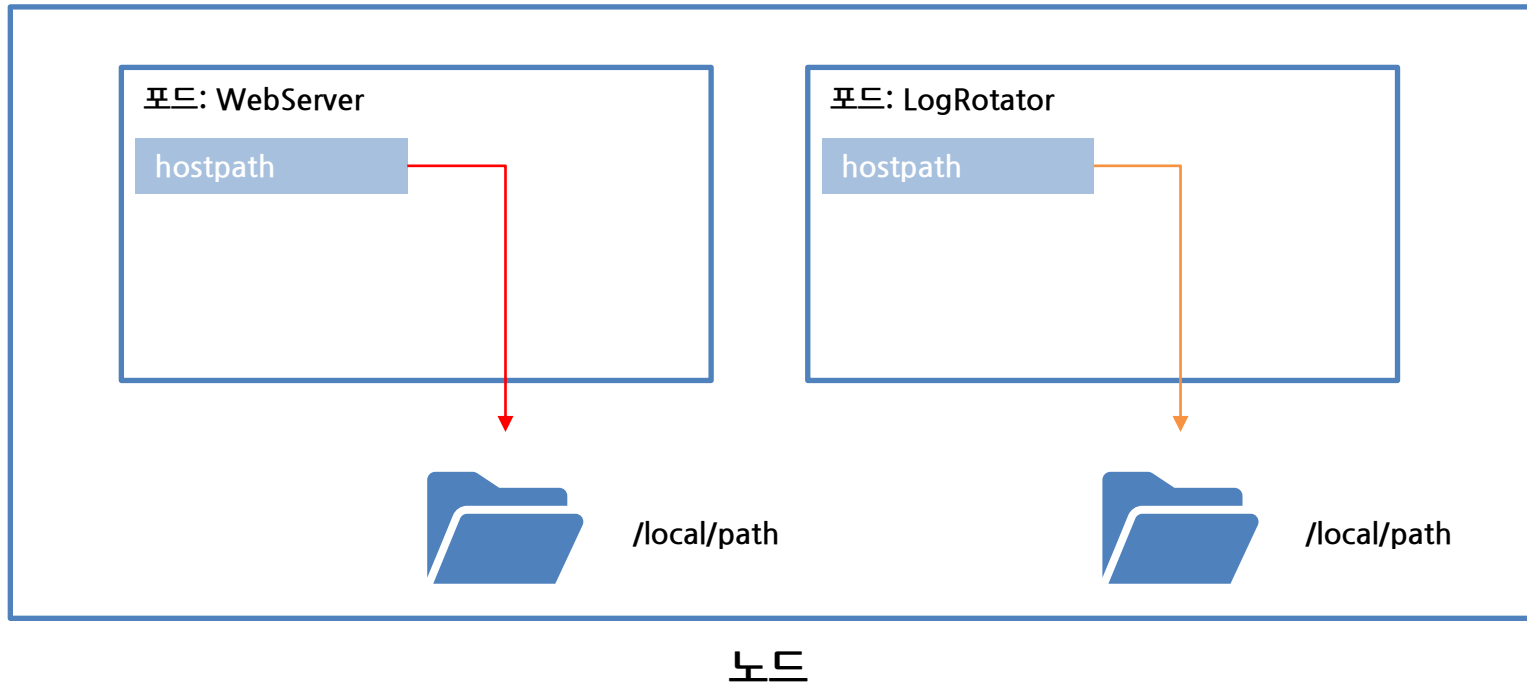
▶▶ 연습문제

- 같은 emptyDir을 공유하는 프로세스를 만들어 통신하기
 - 하나의 프로세스는 랜덤한 값을 생성 (count 이미지 사용)
 - 하나의 프로세스는 랜덤한 값을 출력 (httpd 이미지 사용)

Storage

hostPath 볼륨

- 노드의 파일 시스템에 있는 특정 파일 또는 디렉터리 지정
- 영구 스토리지
- 다른 노드의 포드끼리 데이터 공유는 안됨



Storage

hostPath 볼륨

● hostPath 사용 현황 파악하기

```
$ kubectl get pods --namespace kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
event-exporter-v0.2.4-5f7d5d7dd4-86zvt	2/2	Running	0	47h
fluentd-gcp-scaler-7b895cbc89-6w26t	1/1	Running	0	47h
fluentd-gcp-v3.2.0-fgc7k	2/2	Running	0	46h
fluentd-gcp-v3.2.0-lj6zg	2/2	Running	0	46h
[...]				

```
$ kubectl describe pod fluentd-gcp-v3.2.0-fgc7k --namespace kube-system
```

Name: fluentd-gcp-v3.2.0-fgc7k

Namespace: kube-system

[...]

Volumes:

varlog:

Type: HostPath (bare host directory volume)

Path: /var/log

HostPathType:

varlibdockercontainers:

Type: HostPath (bare host directory volume)

Path: /var/lib/docker/containers

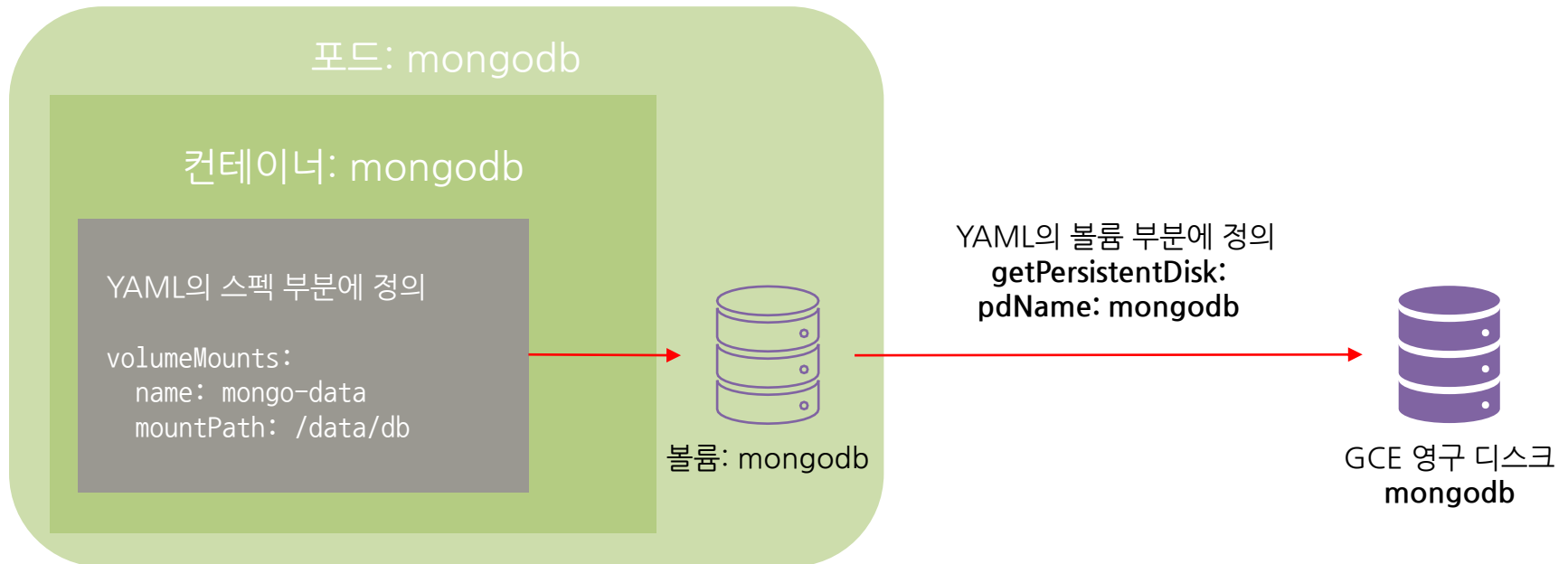
HostPathType:

[중략]

Storage

▶ GCE 영구 스토리지 사용하기

- 데이터가 지속
- 모든 포드에서 공유
- mongo 컨테이너에 마운트하여 사용



Storage

▶ GCE 영구 스토리지 사용하기

- GCE 영구 디스크를 동일한 리전에 생성

```
$ gcloud compute disks create --size=10GiB --zone=us-central1-a mongodb
```

WARNING: You have selected a disk size of under [200GB]. This may result in poor I/O performance.

For more information, see: <https://developers.google.com/compute/docs/disks#performance>.

Created [<https://www.googleapis.com/compute/v1/projects/gketest-244306/zones/us-central1-a/disks/mongodb>].

NAME	ZONE	SIZE_GB	TYPE	STATUS
mongodb	us-central1-a	10	pd-standard	READY

New disks are unformatted. You must format and mount a disk before it can be used. You can find instructions on how to do this at:

<https://cloud.google.com/compute/docs/disks/add-persistent-disk#formatting>

Storage

▶ GCE 영구 스토리지 사용하기

● 포드 YAML의 gcePersistentDisk 정의하기

- ▶ 볼륨만을 별도로 선언할 수 없기 때문에 포드를 생성하여 volume 구성

볼륨에는 mongo-data를 선언하고 gce 디스크를 사용 정의
“pdName”의 명칭은 앞서 생성한 디스크의 이름과 동일해야 함

포드의 어느 디렉토리에 마운트할 지를 결정

mongo-pod-gce.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  volumes:
  - name: mongodb-data
    gcePersistentDisk:
      pdName: mongodb
      fsType: ext4
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
  ports:
  - containerPort: 27017
    protocol: TCP
```

Storage

▶ GCE 영구 스토리지 사용하기

- MongoDB 데이터 입출력 테스트 (입력)

- ▶ YAML파일로 pod를 생성하고 DB가 잘 동작하는 지 확인

```
$ kubectl exec -it mongodb mongo
```

```
MongoDB shell version v4.0.10
```

```
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb  
[...]
```

```
> use mystore
```

```
switched to db mystore
```

```
> db.foo.insert({name:'test',value:'1234'})
```

```
WriteResult({ "nInserted" : 1 })
```

```
> db.foo.find()
```

```
{ "_id" : ObjectId("5d0f0b03a3edc611a05bda93"), "name" : "foo", "value" : "1234" }
```


Storage

▶▶ NFS 네트워크 볼륨 사용하기

● NFS 네트워크 볼륨이 있어야 K8S와 테스트 가능

- 서버 설치 방법
 - ✓ apt-get update
 - ✓ apt-get install nfs-common nfs-kernel-server portmap
- 공유할 디렉토리 생성
 - ✓ mkdir /home/nfs
 - ✓ chmod 777 /home/nfs
- /etc/exports 파일에 다음 내용 추가
 - ✓ /home/nfs 10.0.2.15(rw,sync,no_subtree_check) 10.0.2.4(rw,sync,no_subtree_check)
10.0.2.5(rw,sync,no_subtree_check)
 - ✓ service nfs-server restart
 - ✓ showmount -e 127.0.0.1
- NFS 클라이언트에서는 mount 명령어로 마운트해서 사용
 - ✓ mount -t nfs 10.0.2.5:/home/nfs /mnt

Storage

▶ NFS 네트워크 볼륨 사용하기

- 각 노드에 NFS 관련 라이브러리 설치
 - apt-get update
 - apt-get install nfs-common nfs-kernel-server portmap
- /home/nfs에 index.html을 생성
- nfs-httpd.yaml 파일을 실행 후 접속 테스트

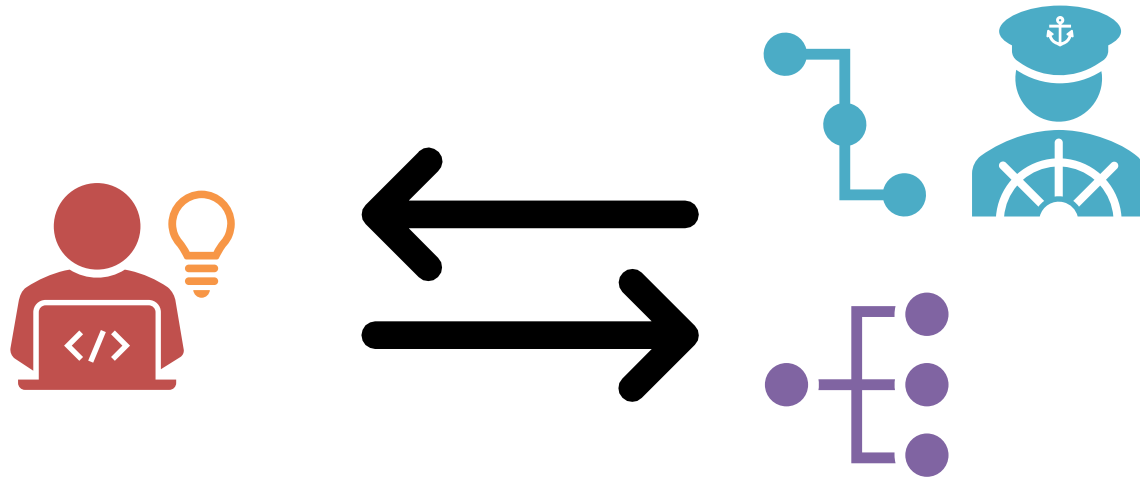
nfs-httpd.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-httpd
spec:
  containers:
    - image: httpd
      name: web
      volumeMounts:
        - mountPath: /usr/local/apache2/htdocs
          name: nfs-volume
          readOnly: true
  volumes:
    - name: nfs-volume
      nfs:
        server: 10.0.2.5
        path: /home/nfs
```

Storage

▶ 포드 개발자 입장에서의 추상화

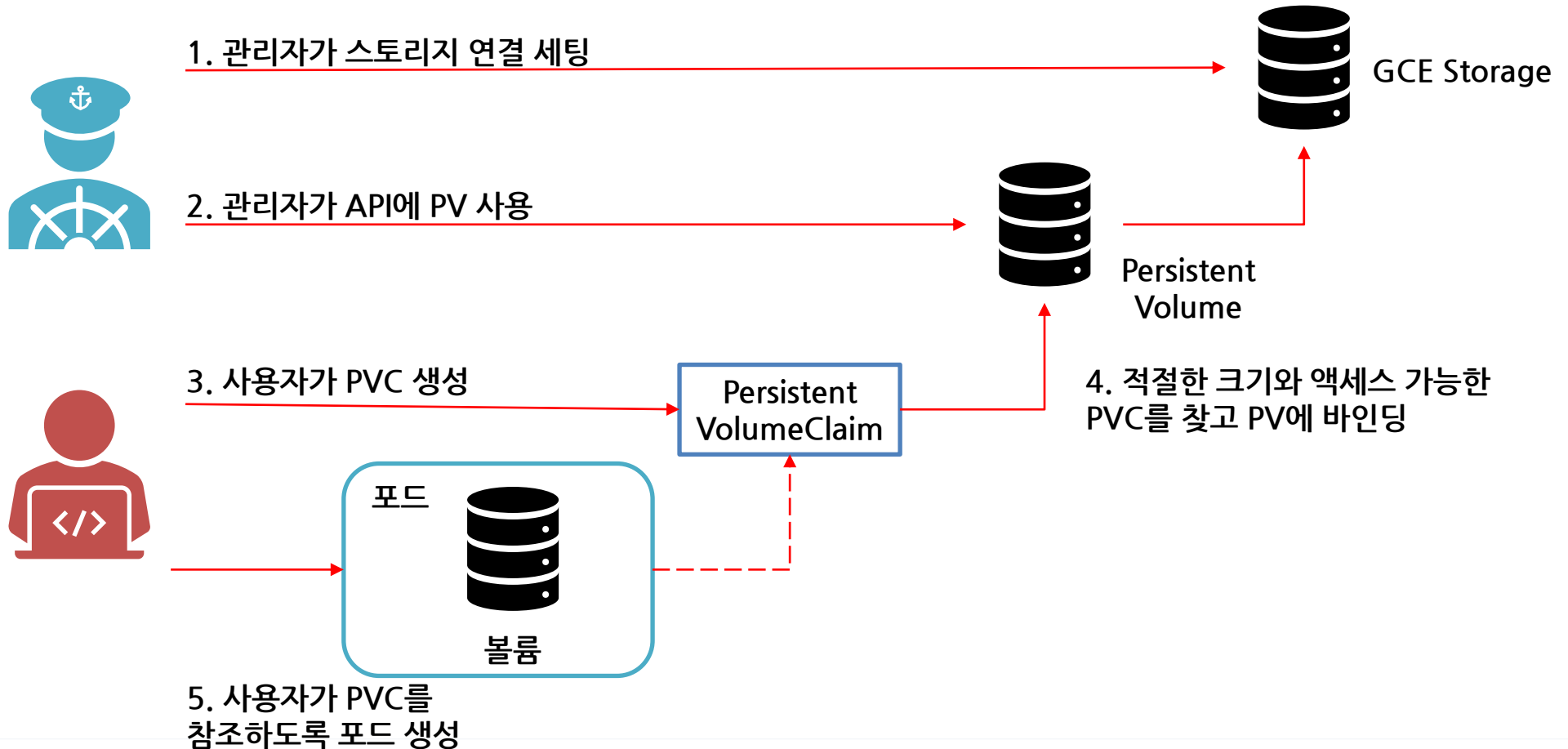
- 포드 개발자가 클러스터에서 스토리지를 사용할 때 인프라를 알아야 할까?
- 실제 네트워크 스토리지를 사용하려면 알아야 함
- 애플리케이션을 배포하는 개발자가 스토리지 기술의 종류를 몰라도 상관없도록 하는 것이 이상적
- 인프라 관련 처리는 클러스터 관리자의 유일한 도메인!
- pv와 pvc를 사용해 관리자와 사용자의 영역을 나눔



Storage

▶ PersistentVolume(PV)과 PersistentVolumeClaim(PVC)

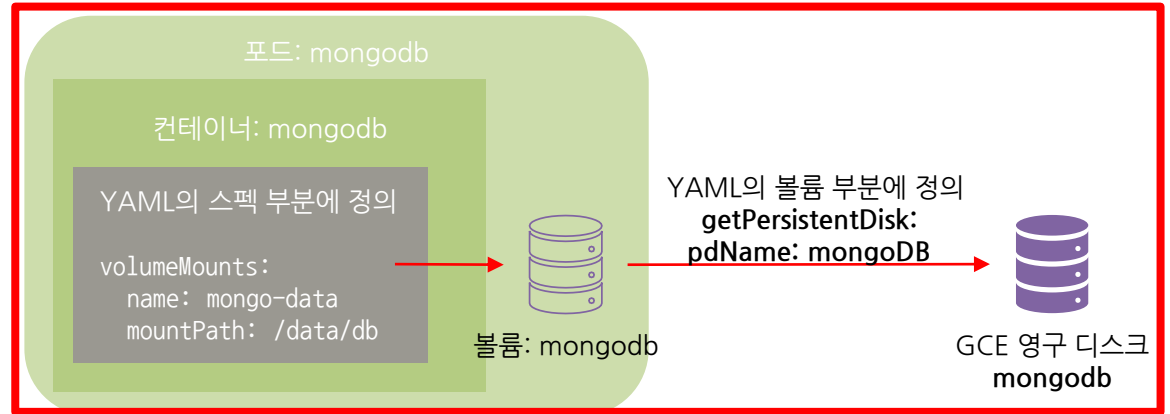
- 인프라 세부 사항을 알지 못해도 클러스터의 스토리지를 사용할 수 있도록 제공하는 리소스
- 포드 안에 영구 볼륨을 사용하도록 하는 방법은 다소 복잡



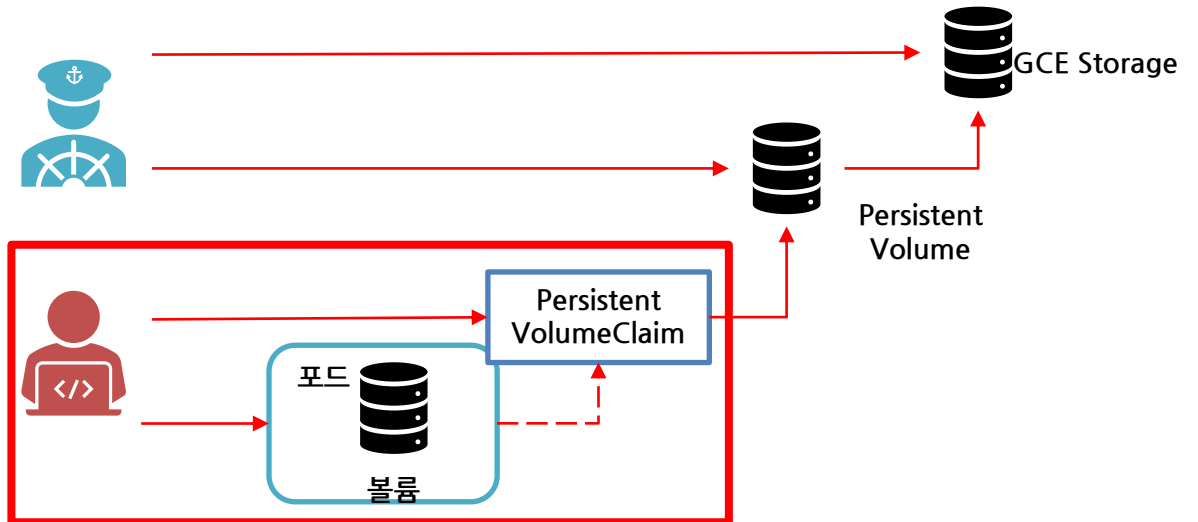
Storage

PV, PVC의 장점 비교

getPersistentDisk를 사용할 때
사용자가 알아야 하는 부분



pv, pvc를 사용할 때
사용자가 알아야 하는 부분



Storage

▶ PersistentVolume(PV)과 PersistentVolumeClaim(PVC) 정의

- 두 가지 새로운 요소에 대해 정의: PVC

mongo-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: ""
```

클레임 사용 때 필요

요청하는 스토리지 양

단일 클라이언트에
읽기쓰기 지원

동적 프로비저닝에서 사용

Storage

▶ PersistentVolume(PV)과 PersistentVolumeClaim(PVC) 정의

● 두 가지 새로운 요소에 대해 정의: PV

- 참고: PV는 네임스페이스에 속하지 않는다!

mongo-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  gcePersistentDisk:
    pdName: mongodb
    fsType: ext4
```

몽고DB에 대한 정의

단일 클라이언트에 읽기쓰기 가능
여러 번 읽기만 가능

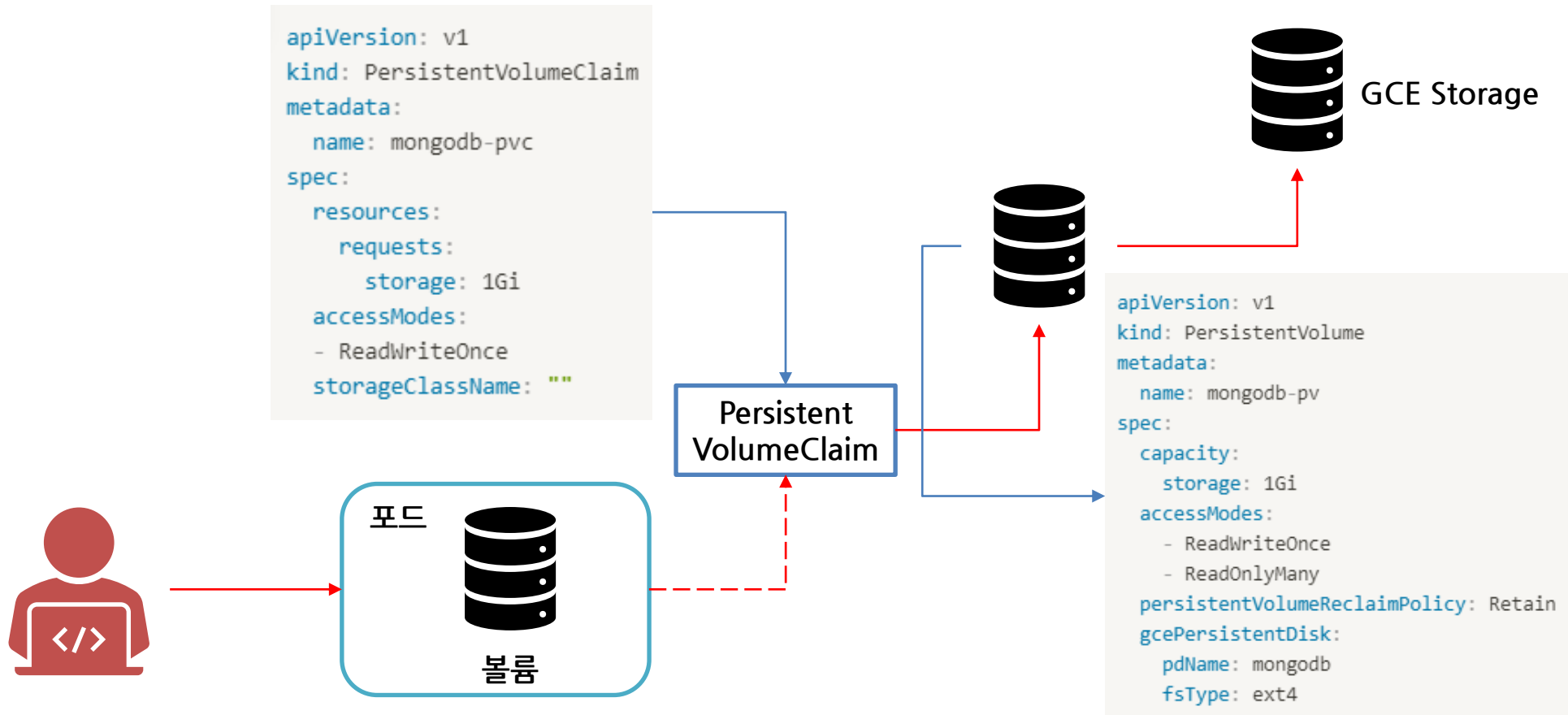
Reclaiming	설명
Retain(유지)	PersistentVolumeClaim삭제하면 PersistentVolume여전히 존재하고 볼륨은 "해제된"것으로 간주 연관된 스토리지 자산의 데이터를 수동으로 정리
Delete(삭제)	외부 인프라의 연관된 스토리지 자산을 모두 제거
Recycle(재사용)	rm -rf /thevolume/*볼륨에 대한 기본 스크립트()을 수행 하고 새 클레임에 대해 다시 사용할 수 있도록 함

※ 설정은 생성 후에도 재설정 가능

Storage

▶ PersistentVolume(PV)과 PersistentVolumeClaim(PVC)

- 인프라 세부 사항을 알지 못해도 클러스터의 스토리지를 사용할 수 있도록 제공하는 리소스
- 포드 안에 영구 볼륨을 사용하도록 하는 방법은 다소 복잡



Storage

PV, PVC 생성과 조회

```
$ kubectl delete all --all
$ kubectl create -f mongo-pv.yaml
$ kubectl create -f mongo-pvc.yaml
```

```
$ kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
mongodb-pvc	Bound	mongodb-pv	1Gi	RWO,RX		30s

```
$ kubectl get pv
```

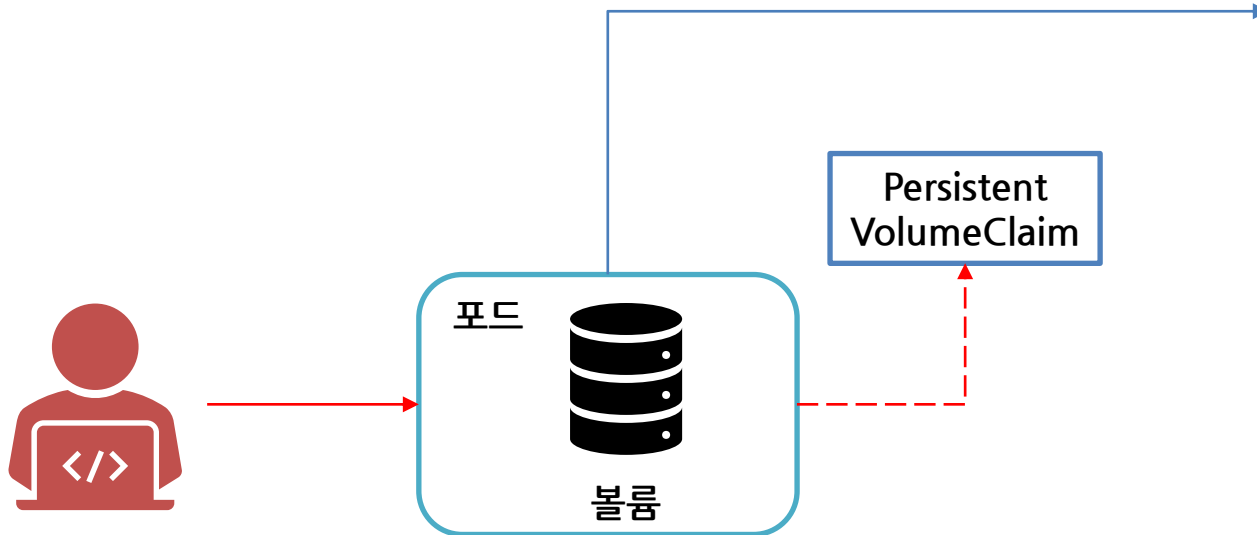
NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	...
mongodb-pv	1Gi	RWO,RX	Retain	Bound	default/mongodb-pvc	...

Storage

PVC를 활용한 포드 생성

```
$ kubectl create -f mongo-pvc-pod.yaml  
pod/mongodb created
```

```
$ kubectl exec -it mongodb mongo  
MongoDB shell version v4.0.10  
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb  
[...]  
> use mystore  
switched to db mystore  
> db.foo.find()  
{ "_id" : ObjectId("5d0f0b03a3edc611a05bda93"), "name" : "foo" }
```



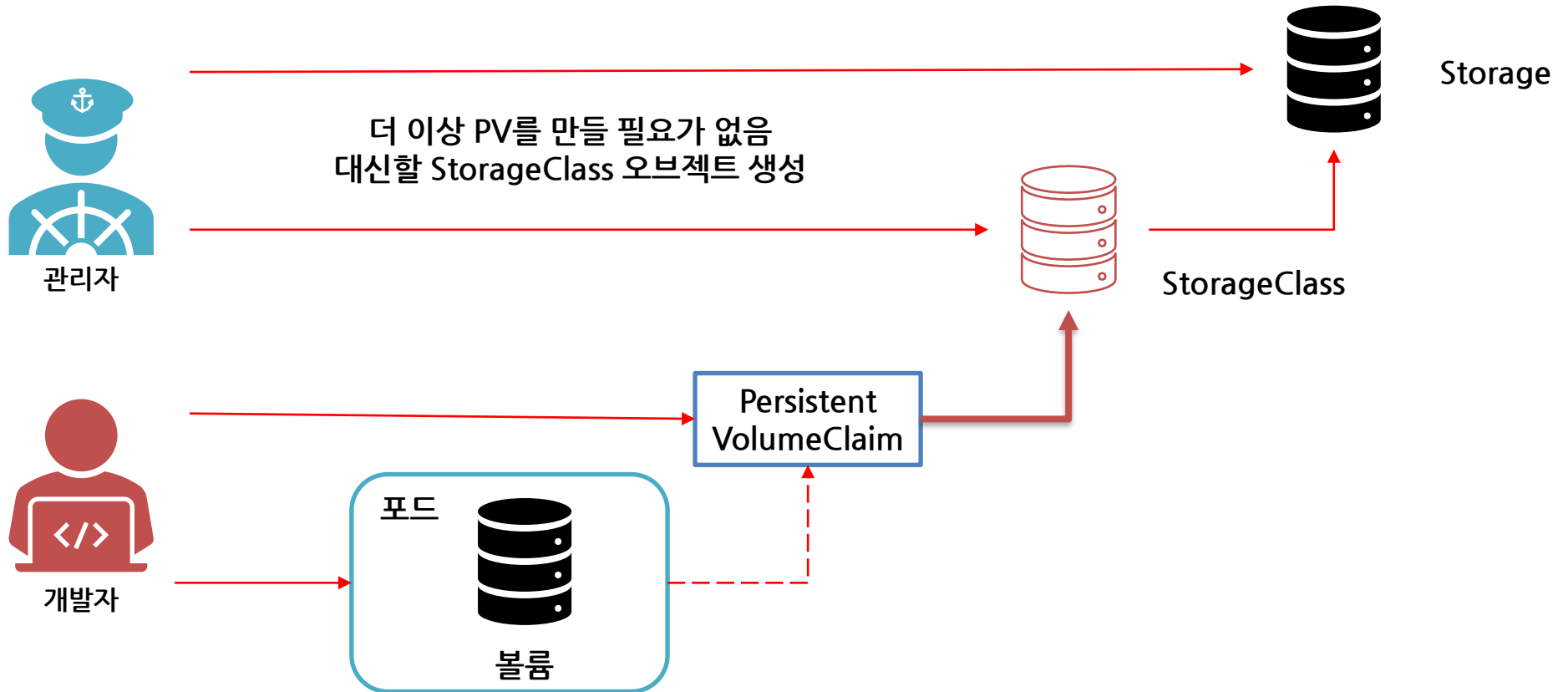
mongo-pvc-pod.yaml

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: mongodb  
spec:  
  containers:  
  - image: mongo  
    name: mongodb  
    volumeMounts:  
    - name: mongodb-data  
      mountPath: /data/db  
    ports:  
    - containerPort: 27017  
      protocol: TCP  
  volumes:  
  - name: mongodb-data  
    persistentVolumeClaim:  
      claimName: mongodb-pvc
```

Storage

PV 동적 프로비저닝

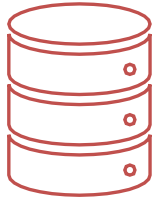
- PV를 직접 만드는 대신 사용자가 원하는 PV 유형을 선택하도록 오브젝트 정의 가능



Storage

▶ PV 동적 프로비저닝

● StorageClass yaml 파일 제작



storageclass.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

프로비저닝에 사용할 플러그인 선택

제공자에게 전달될 매개 변수

```
$ kubectl create -f storageclass.yaml
storageclass.storage.k8s.io/storage-class created
```

```
$ kubectl get sc
```

NAME	PROVISIONER	AGE
standard (default)	kubernetes.io/gce-pd	2d7h
storage-class	kubernetes.io/gce-pd	26s

Storage

▶ PV 동적 프로비저닝

● PVC 파일 제작

- 포드와 PVC 모두 삭제 후 재 업로드 (apply 명령어 시 권한 에러 발생)
- mongo-pvc.yaml 내용 변경

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: ""
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: storage-class
```



Storage

PV 동적 프로비저닝

- PV 동적 프로비저닝을 사용하면 사용할 디스크와 PV가 자동으로 생성됨

```
$ gcloud compute disks list
```

NAME	...	SIZE_GB	TYPE	STATUS
...	... 20	pd-standard	READY	
...	... 20	pd-standard	READY	
gke-standard-cluster-1-default-pool-e4bbb8da-9th0	...	20	pd-standard	READY
...				
gke-standard-cluster-1-pvc-c285a414-95c7-11e9-81f9-42010a9200df	...	1	pd-ssd	READY

```
$ kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM
STORAGECLASS REASON AGE					
mongodb-pv	1Gi	RWO,ROX	Retain	Released	
default/mongodb-pvc	38m				
pvc-c285a414-95c7-11e9-81f9-42010a9200df	1Gi	RWO	Delete	Bound	
default/mongodb-pvc storage-class	15m				

Storage

▶ PV 동적 프로비저닝 동작 순서 정리

mongo-pvc-pod.yaml (변경사항 없음)

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
  ports:
  - containerPort: 27017
    protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim:
      claimName: mongodb-pvc
```

mongo-pvc.yaml (storageclass 이름 변경)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi
  accessModes:
  - ReadWriteOnce
  storageClassName: storage-class
```

storageclass.yaml (새로 생성)

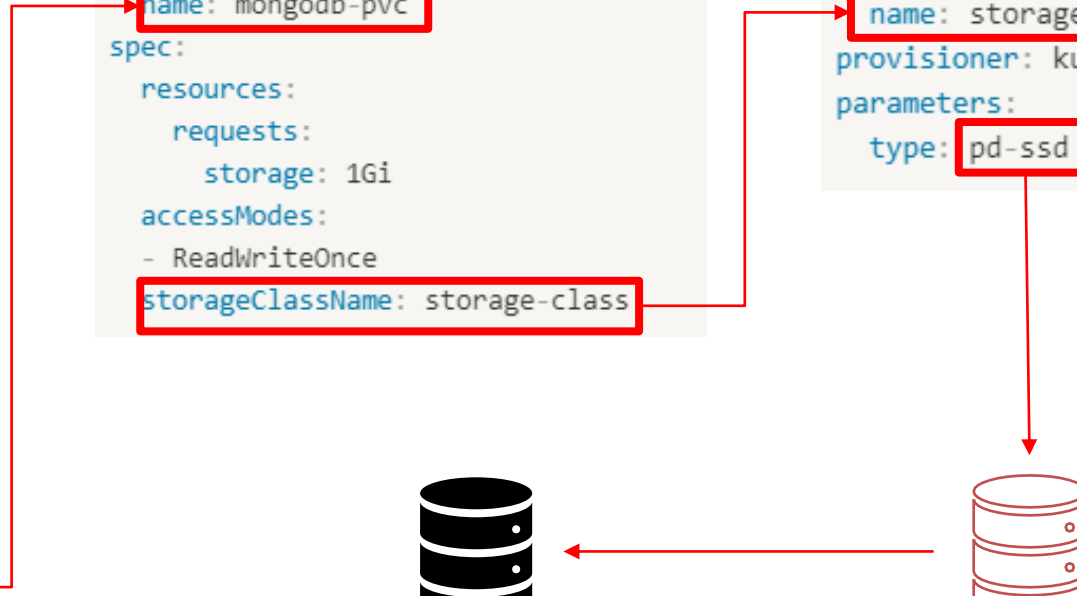
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```



GCE Storage
(자동 생성)



PV (자동 생성)



Storage

▶▶ 연습문제

- mongo를 사용할 수 있도록 pod, pvc, pv 정의하여 수동 프로비저닝 수행하기
 - Mongo를 사용할 수 있도록 수동으로 pod, pvc, pv, disk를 정의하고 생성하라.
- Mongo를 사용할 수 있도록 pod, pvc, storageclass 정의하여 동적 프로비저닝 수행하기
 - Mongo를 사용할 수 있도록 수동으로 pod, pvc, storageclass를 정의하고 생성하라.