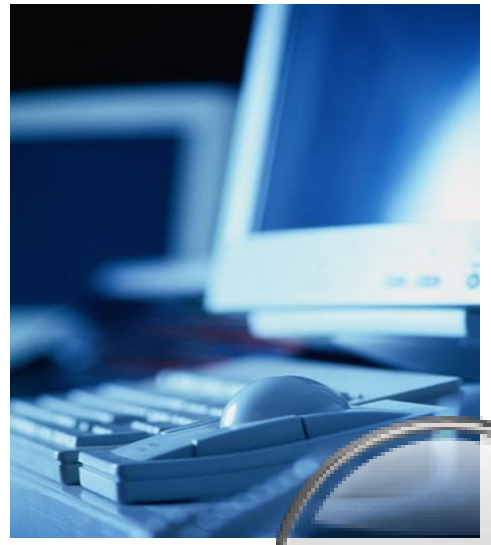
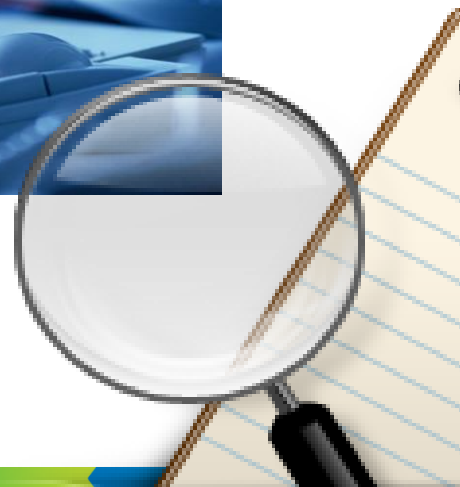
쿠버네티스 저장소

- 볼륨 개요
- Secrets, configmap 마운트
- NFS를 활용한 네트워크 스토리지
- PV와 PVC
- StorageClass
- rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스
- 스테이트풀셋





볼륨 개요

볼륨 개요

볼륨(Volume)

- 컨테이너가 외부 스토리지에 액세스하고 공유하는 방법
- 파드의 각 컨테이너에는 고유의 분리된 파일 시스템 존재
- 볼륨은 파드의 컴포넌트이며 파드의 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 …

볼륨 개요

주요 사용 가능한 볼륨의 유형

- **emptyDir:** 일시적인 데이터 저장, 비어 있는 디렉터리
- **hostPath:** 파드에 호스트 노드의 파일 시스템에서 파일이나 디렉토리를 마운트
- **nfs:** 기존 NFS (네트워크 파일 시스템) 공유가 파드에 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

- 볼륨 관련 레퍼런스
  - https://kubernetes.io/docs/concepts/storage/volumes/#persistentvolumeclaim

Secrets, configmap 마운트

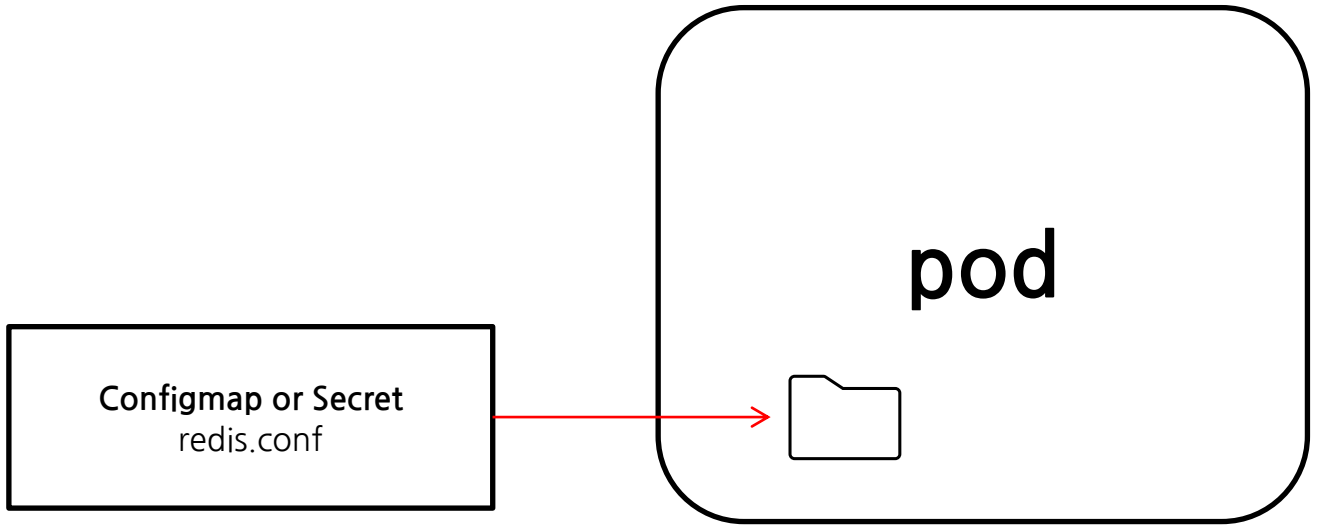
Secrets, configmap 마운트

컨피그맵(ConfigMap) 마운트

- 컨피그맵은 키-값 쌍으로 기밀이 아닌 데이터를 저장하는 데 사용하는 API 오브젝트
- 파드는 볼륨에서 환경 변수, 커맨드-라인 인수 또는 구성 파일로 컨피그맵을 사용
- 컨피그맵을 사용하면 컨테이너 이미지에서 환경별 구성을 분리하여, 애플리케이션을 쉽게 이식

시크릿(Secret) 마운트

- 시크릿은 암호, 토큰 또는 키와 같은 소량의 중요한 데이터를 포함하는 오브젝트
- 시크릿을 사용해 사용자의 기밀 데이터를 애플리케이션 코드에 넣을 필요가 없음



Secrets, configmap 마운트

컨피그맵(ConfigMap) 마운트

- 컨피그맵에 원하는 내용을 넣어서 파일을 구성

```
cat <<EOF >./example-redis-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-redis-config
data:
  redis-config: |
    maxmemory 2mb
    maxmemory-policy allkeys-lru
EOF

kubectl apply -f
https://raw.githubusercontent.com/kubernetes/website/main/content/en/examples/pod
s/config/redis-pod.yaml
```

Secrets, configmap 마운트

컨피그맵(ConfigMap) 마운트

- 컨피그맵이 적절히 전달 됐는지 확인

```
$ kubectl exec -it redis -- redis-cli

127.0.0.1:6379> CONFIG GET maxmemory-policy
1) "maxmemory-policy"
2) "allkeys-lru"

127.0.0.1:6379> CONFIG GET maxmemory
1) "maxmemory"
2) "2097152"
```

Secrets, configmap 마운트

시크릿(Secret) 마운트

- 시크릿에 전달할 데이터와 시크릿 생성

```
echo -n admin > username
echo -n 1q2w3e > password

kubectl create secret generic mysecret --from-file=username --from-file=password
```

Secrets, configmap 마운트

시크릿(Secret) 마운트

- Secret을 통해서 파드에 마운트

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
EOF
```

NFS를 활용한 네트워크 스토리지

NFS를 활용한 네트워크 스토리지

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

NFS를 활용한 네트워크 스토리지

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

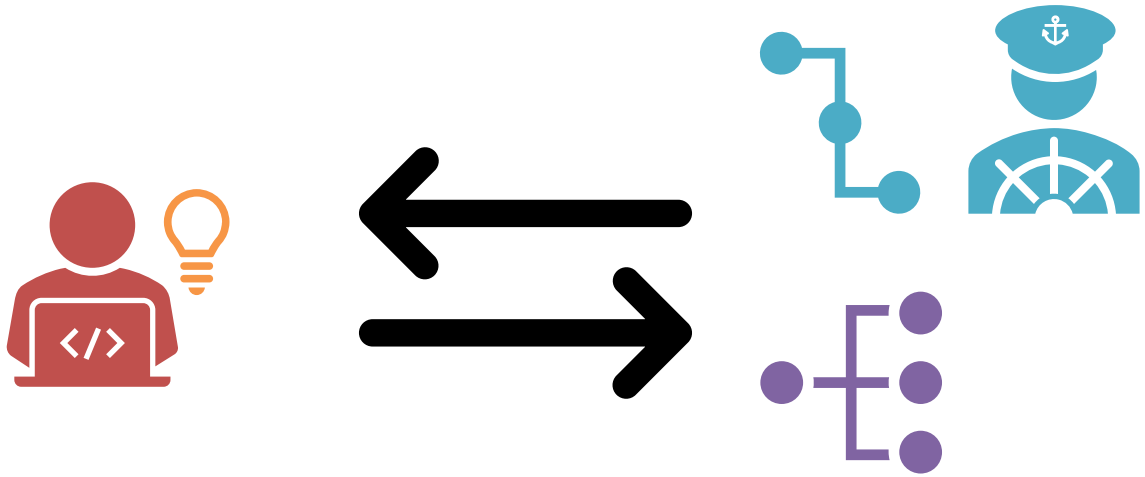
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nfs-httpd
spec:
  containers:
  - image: httpd
    name: web
    volumeMounts:
    - mountPath: /usr/local/apache2/htdocs
      name: nfs-volume
      readOnly: true
  volumes:
  - name:  nfs-volume
    nfs:
      server: 10.0.2.5
      path: /home/nfs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

PV와 PVC

PV와 PVC

파드 개발자 입장에서의 추상화

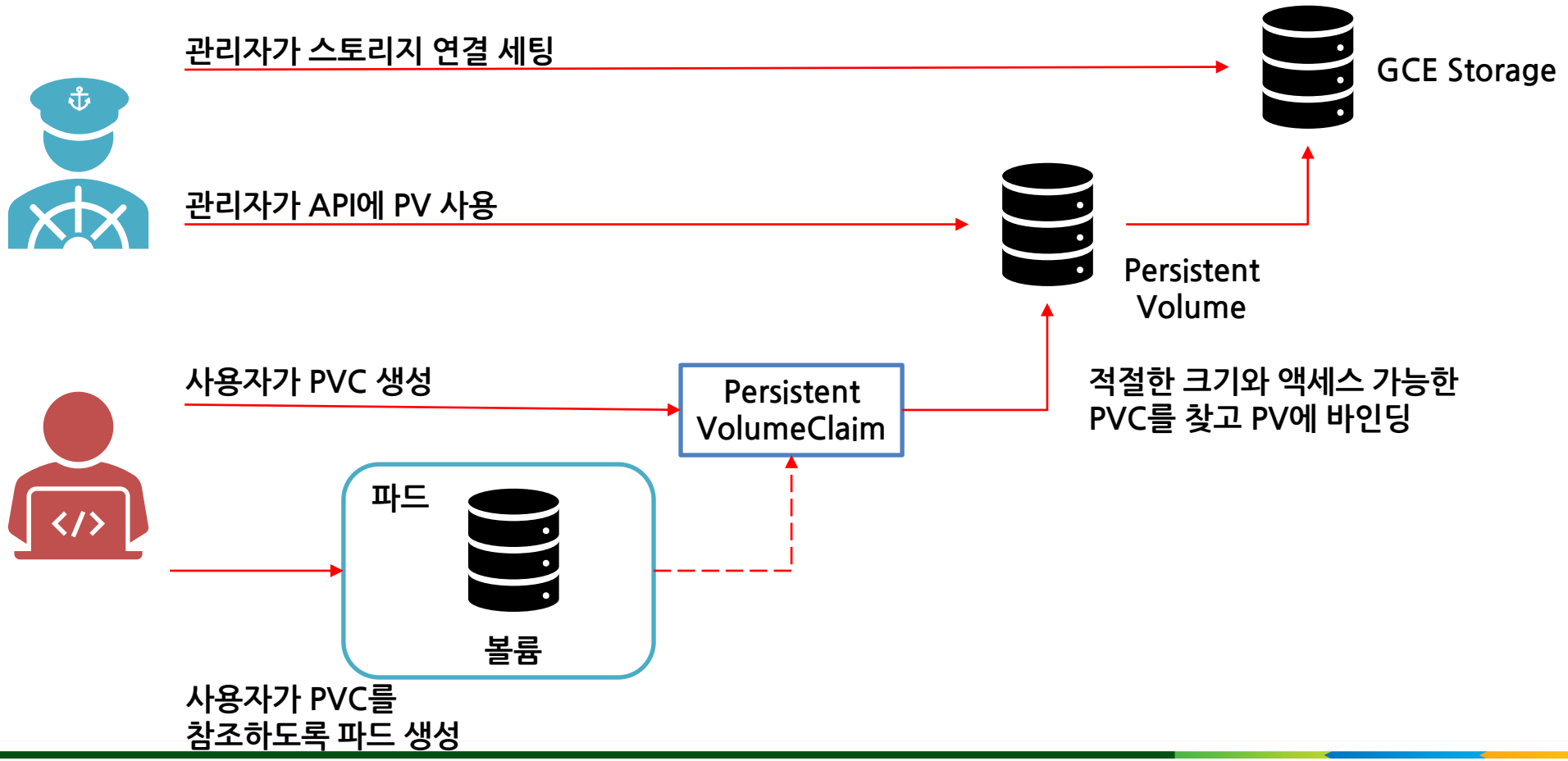
- 파드 개발자가 클러스터에서 스토리지를 사용할 때 인프라를 알아야 할까?
- 실제 네트워크 스토리지를 사용하려면 알아야 함
- 애플리케이션을 배포하는 개발자가 스토리지 기술의 종류를 몰라도 상관없도록 하는 것이 이상적
- 인프라 관련 처리는 클러스터 관리자의 유일한 도메인!
- pv와 pvc를 사용해 관리자와 사용자의 영역을 나눔



PV와 PVC

PersistentVolume(PV)과 PersistentVolumeClaim(PVC)

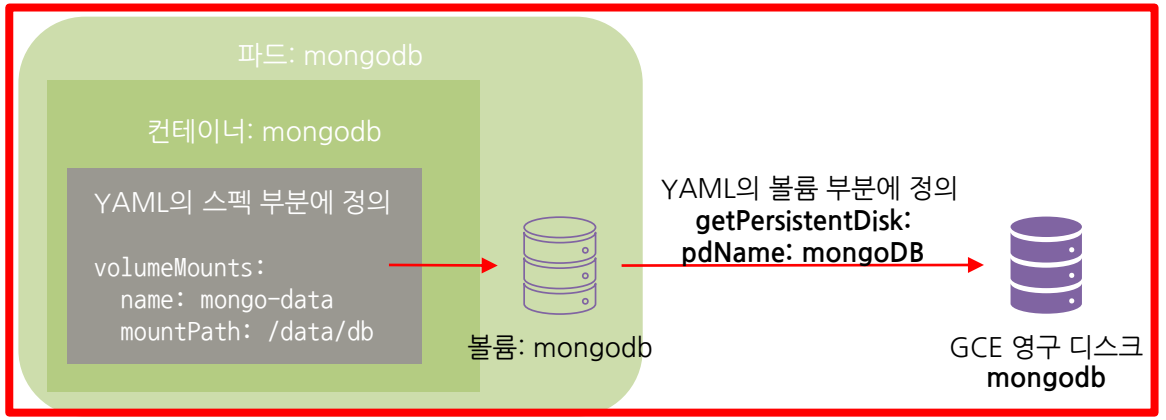
- 인프라 세부 사항을 알지 못해도 클러스터의 스토리지를 사용할 수 있도록 제공해주는 리소스
- 파드 안에 영구 볼륨을 사용하도록 하는 방법은 다소 복잡



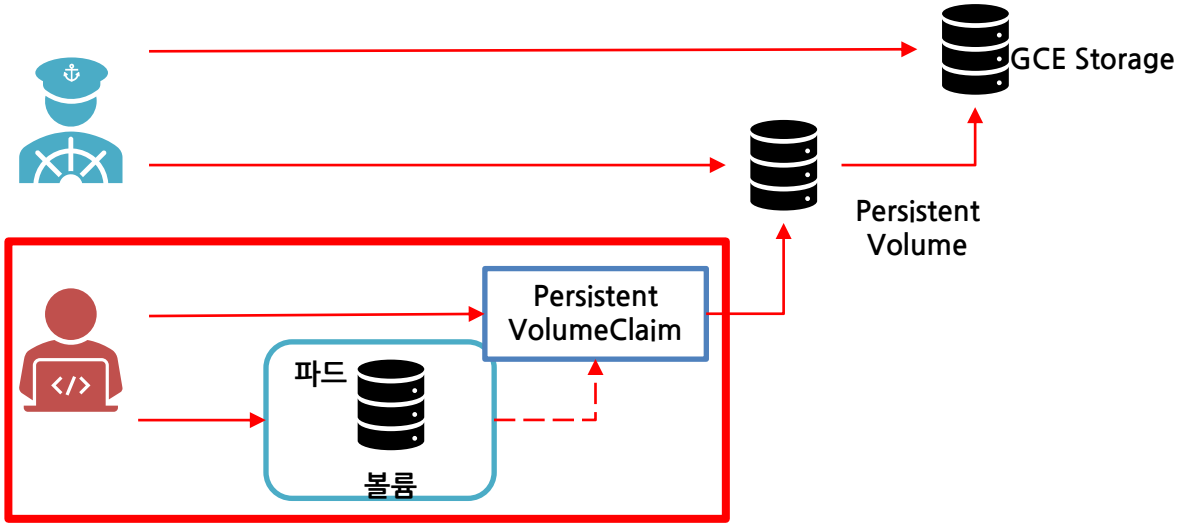
PV와 PVC

PV, PVC의 장점 비교

getPersistentDisk를 사용할 때
사용자가 알아야 하는 부분



pv, pvc를 사용할 때
사용자가 알아야 하는 부분



PV와 PVC

PersistentVolume(PV)과 PersistentVolumeClaim(PVC) 정의

- 두 가지 새로운 요소에 대해 정의: PVC

mongo-pvc.yaml

```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

PV와 PVC

PersistentVolume(PV)과 PersistentVolumeClaim(PVC) 정의

- 두 가지 새로운 요소에 대해 정의: PV
  - 참고: PV는 네임스페이스에 속하지 않는다!

mongo-pv.yaml

```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

Reclaming	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 스크럽()을 수행 하고 새 클레임에 대해 다시 사용할 수 있도록 함

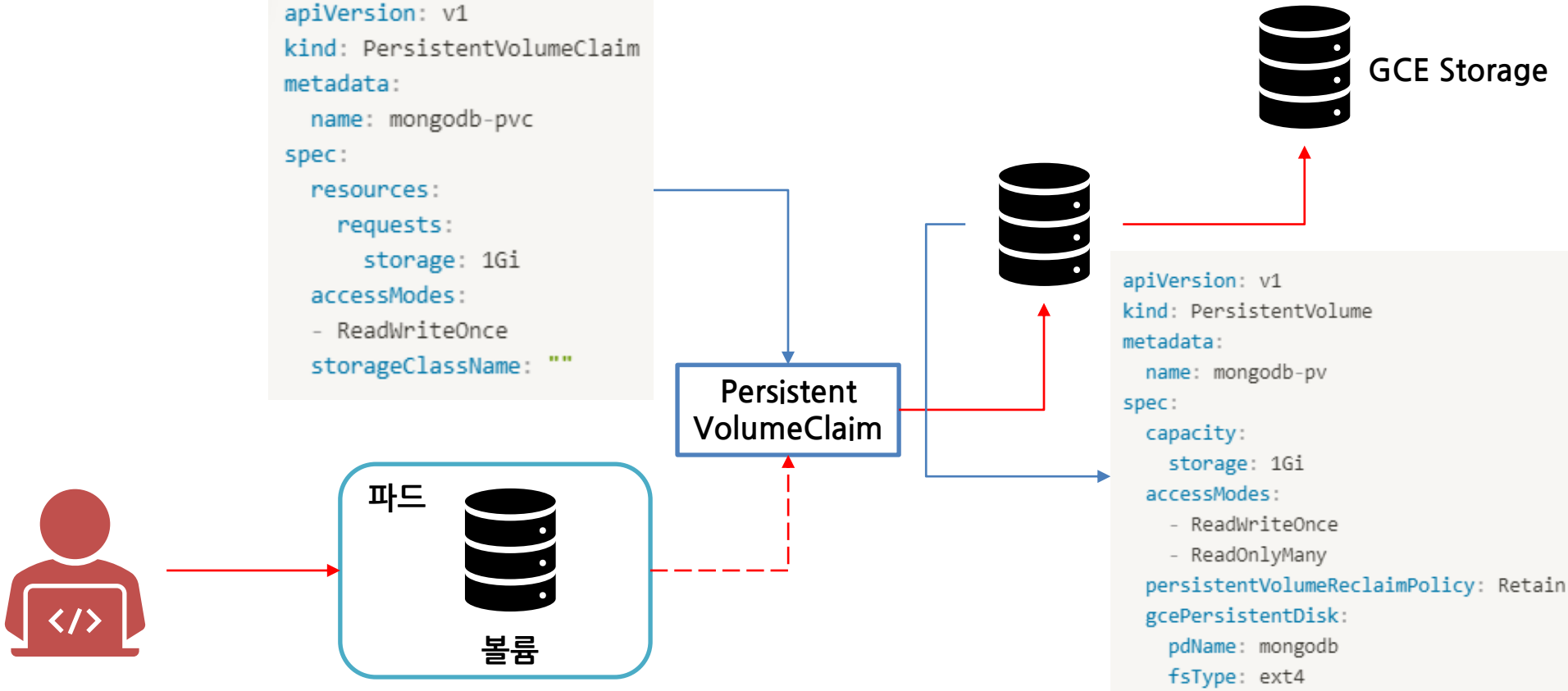
※ 설정은 생성 후에도 재설정 가능

PV와 PVC

PersistentVolume(PV)과 PersistentVolumeClaim(PVC)

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

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




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

PV와 PVC

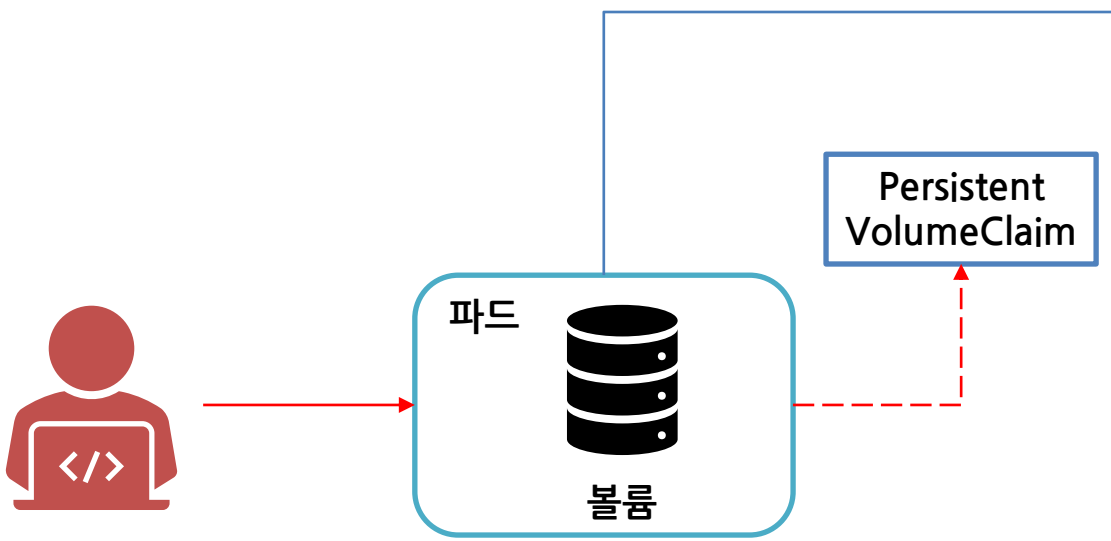
PV, PVC 생성과 조회

```
$ 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,ROX                       30s

$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 …
mongodb-pv   1Gi        RWO,ROX        Retain           Bound    default/mongodb-pvc   …
```

PV와 PVC

PVC를 활용한 파드 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

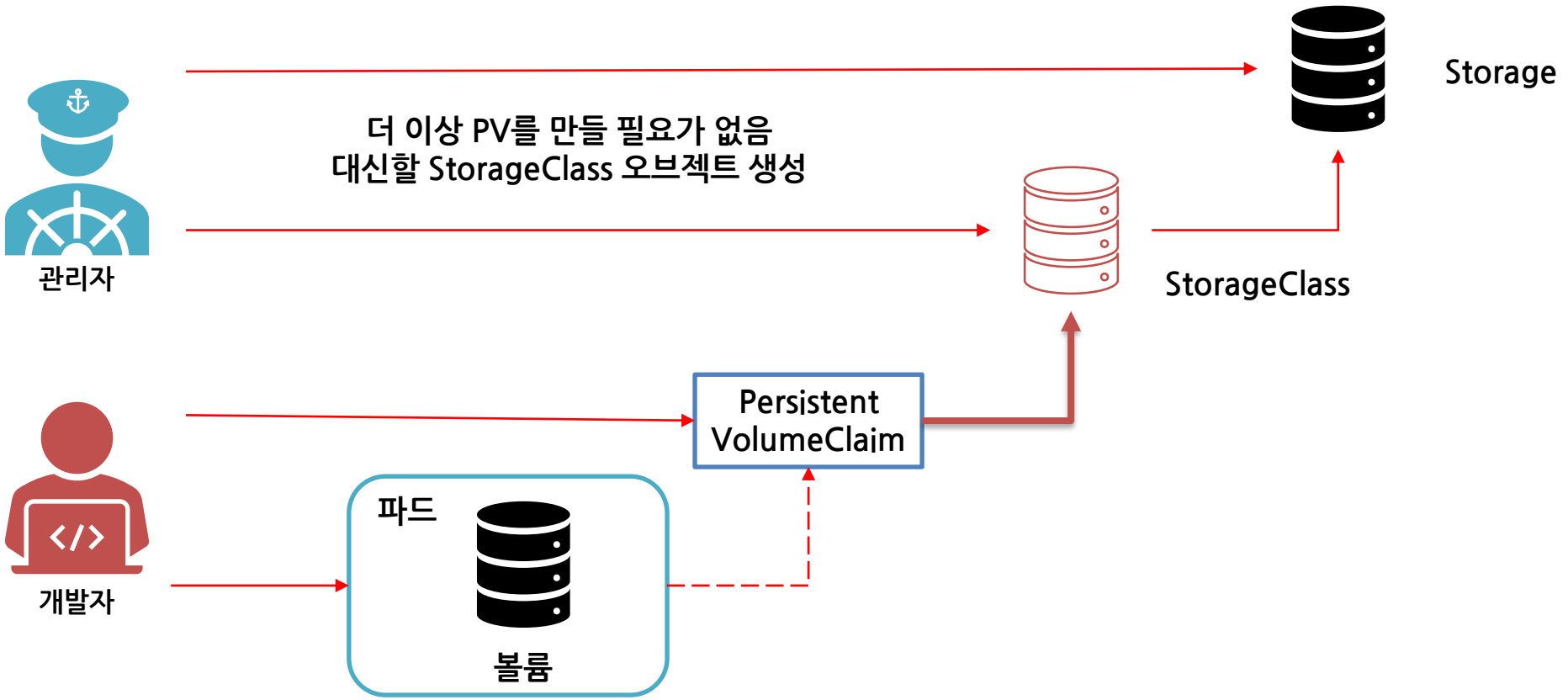
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim: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

StorageClass

StorageClass

PV 동적 프로비저닝

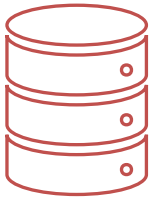
- PV를 직접 만드는 대신 사용자가 원하는 PV 유형을 선택하도록 오브젝트 정의 가능



StorageClass

PV 동적 프로비저닝

- StorageClass yaml 파일 제작



storageclass.yaml

```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
```

PV 동적 프로비저닝

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

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

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

StorageClass

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
```

StorageClass

PV 동적 프로비저닝 동작 순서 정리

mongo-pvc-pod.yaml (변경사항 없음)

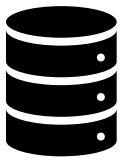
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim: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

mongo-pvc.yaml (storageclass 이름 변경)

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

storageclass.yaml (새로 생성)

```yaml
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)

연습문제

- **httpd를 사용할 수 있도록 pod, pvc, pv 정의하여 수동 프로비저닝 수행하기**
  - httpd를 사용할 수 있도록 수동으로 pod, pvc, pv, disk를 정의하고 생성하라.

- **httpd를 사용할 수 있도록 pod, pvc, storageclass 정의하여 동적 프로비저닝 수행하기**
  - httpd를 사용할 수 있도록 자동으로 pod, pvc, storageclass를 정의하고 생성하라.

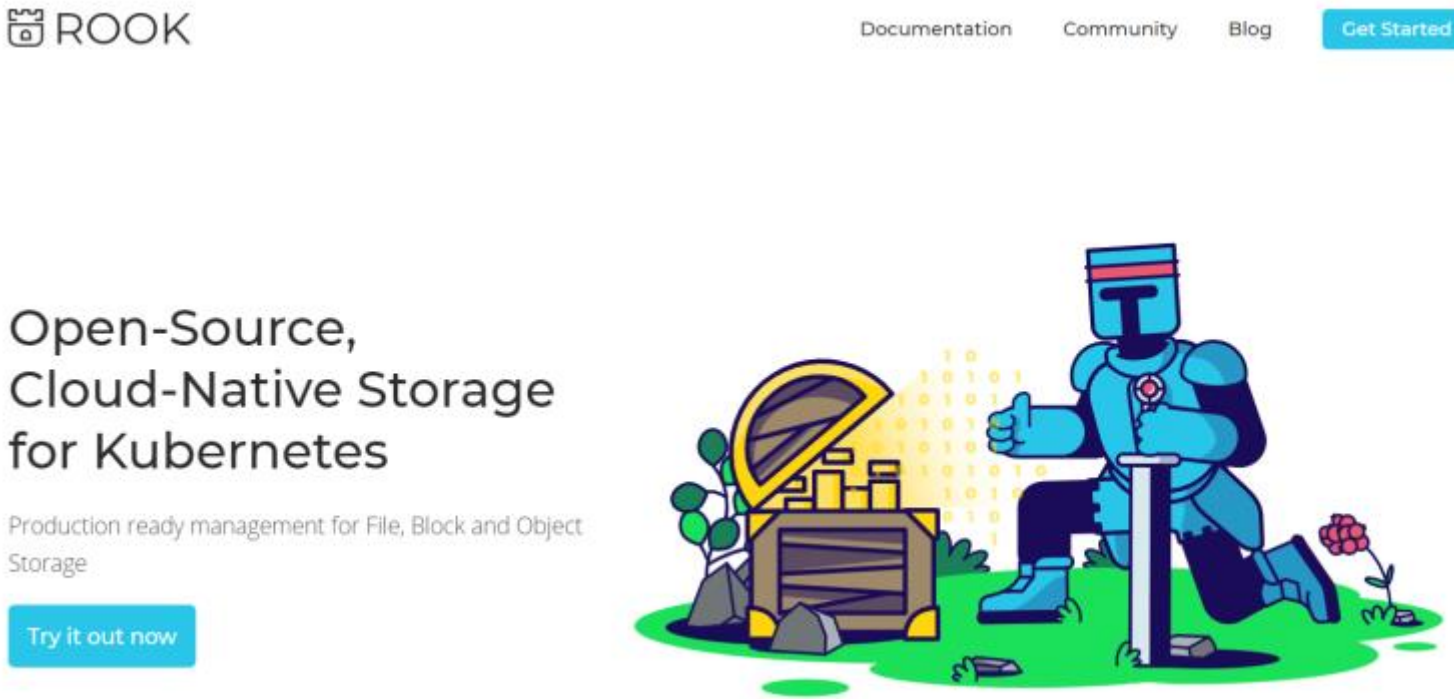
rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

rook-ceph?

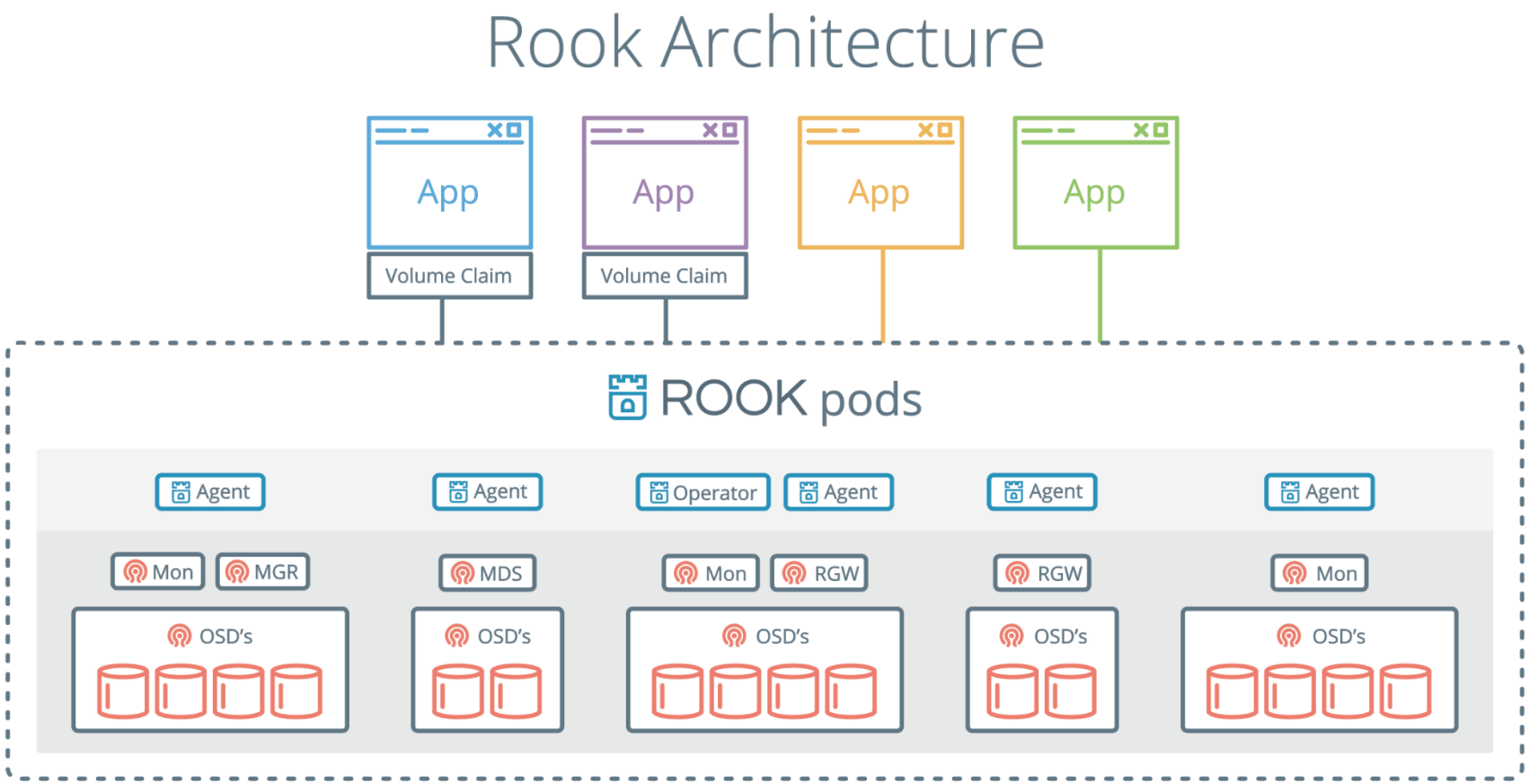
- 온프레미스 환경에서 storage-class를 구성하는 도구
- ceph은 파일 스토리지를 가상화시키는 클러스터를 구성할 수 있는 소프트웨어
- 직접 설치하는 방법도 있지만 rook 패키지를 활용하면 쿠버네티스에서 보다 편리하게 ceph을 설치하고 관리 가능

http://rook.io/



rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

rook-ceph 아키텍처

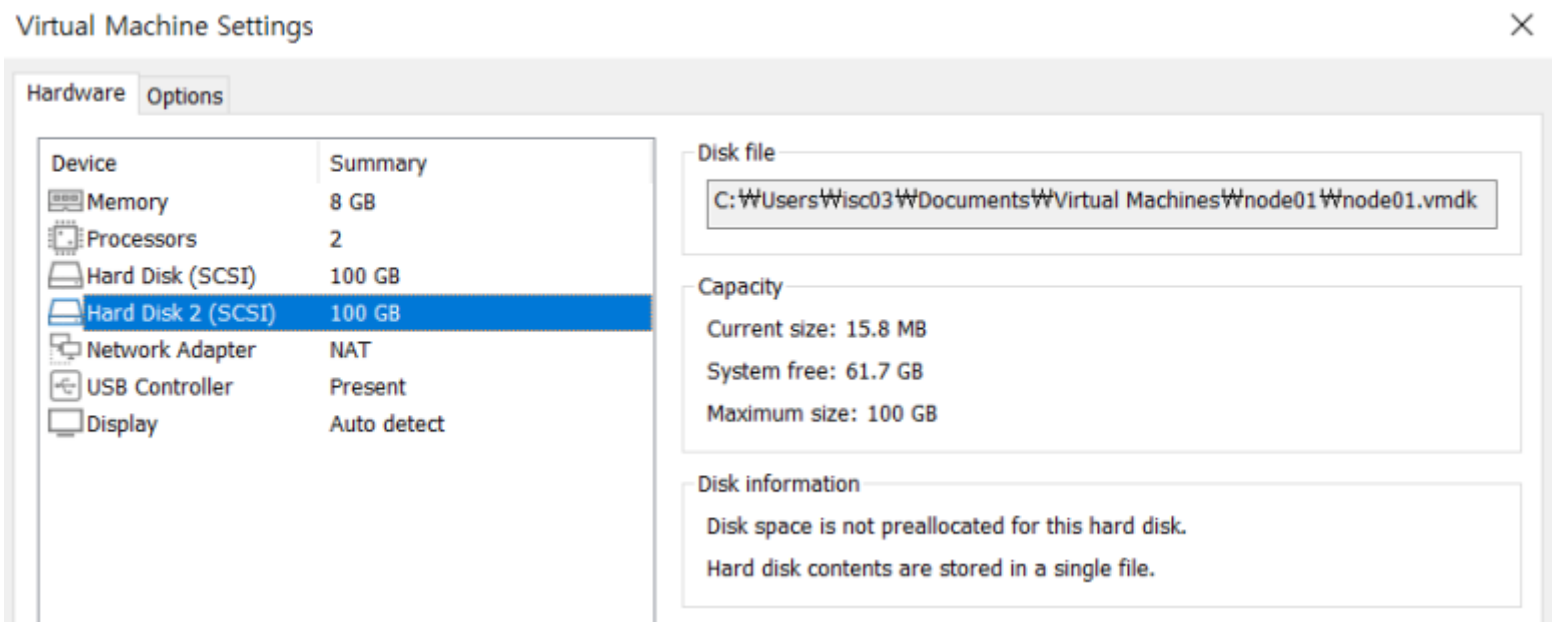


https://danawalab.github.io/kubernetes/2020/01/28/kubernetes-rook-ceph.html

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

설치를 시작하기 전에

- 새 disk 추가 및 lsblk 확인 필요
- 설치를 시작하기 전에 3개의 워커 노드를 준비
- 각 워커 노드에 새로운 빈 디스크를 하나 추가



rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

설치를 시작하기 전에

- 잘 구성하고 lsblk 명령을 실행하면 다음과 같이 빈 디스크를 확인
- 하위에 다른 파티션이 구현되어 있어 있어서는 안됨

```
user01@node01:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0    7:0    0 55.4M  1 loop /snap/core18/2128
loop1    7:1    0   55M  1 loop /snap/core18/1705
loop2    7:2    0 72.5M  1 loop /snap/lxd/21497
loop3    7:3    0 32.3M  1 loop /snap/snapd/12883
loop4    7:4    0   69M  1 loop /snap/lxd/14804
loop5    7:5    0 61.8M  1 loop /snap/core20/1081
loop6    7:6    0 27.1M  1 loop /snap/snapd/7264
sda      8:0    0  100G  0 disk
├─sda1   8:1    0    1M  0 part
└─sda2   8:2    0  100G  0 part /
sdb      8:16   0  100G  0 disk
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

루크로 ceph 설치하기

- 루크를 깃에서 다운로드하고 설치를 시작
- 한 번에 모든 명령을 실행하면 정확히 실행되지 않는 경우들이 있으니 명령어를 하나씩 실행
- rook 프로젝트에는 다양한 기능이 있으나 그중에 kubernetes ceph을 설치

```
git clone --single-branch --branch release-1.7
https://github.com/rook/rook.git
cd rook/cluster/examples/kubernetes/ceph
kubectl create -f crds.yaml -f common.yaml -f operator.yaml
kubectl create -f cluster.yaml
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

툴 박스와 ceph csi를 설치

- 툴박스는 ceph의 상황을 모니터링할 수 있는 툴을 구성한 컨테이너
- csi는 Container Storage Interface라는 의미를 가짐
- csi는 각 노드에 컨테이너에서 스토리지를 사용할 수 있는 인터페이스를 제공

```
kubectl create -f toolbox.yaml
kubectl create -f csi/rbd/storageclass.yaml
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

툴박스 컨테이너로 상태 확인

- 로 접속해서 ceph 스토리지의 status를 확인 가능
- 다음 명령을 실행해 ceph 툴박스 컨테이너로 접속하고 상태 확인

```
$ kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-
tools" -o jsonpath='{.items[0].metadata.name}') -- bash

# 컨테이너 내부에서 다음 명령 실행
ceph status
ceph osd pool stats
```

스토리지 클래스 확인

```
$ kubectl get sc
NAME              PROVISIONER                  RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEX...
rook-ceph-block   rook-ceph.rbd.csi.ceph.com   Delete          Immediate           true
```

rook-ceph를 활용한 프라이빗 클라우드 스토리지클래스

스토리지클래스 pv 생성 테스트

- pvc를 다음과 같이 생성

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  storageClassName: rook-ceph-block
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
EOF
```

- 대응되는 pv가 자동으로 구성되는지 확인

```
$ kubectl get pvc,pv
NAME                                STATUS   VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS      AGE
persistentvolumeclaim/mongo-pvc   Bound    pvc-a5c17990-1c5a-42e5-8642-744a1cefdabd   2Gi        RWO
rook-ceph-block   79m

NAME                                                        CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                 STORAGECLASS      REASON   AGE
persistentvolume/pvc-a5c17990-1c5a-42e5-8642-744a1cefdabd   2Gi        RWO            Delete
Bound    default/mongo-pvc   rook-ceph-block            79m
```

스테이트풀셋

스테이트풀셋

스테이트풀셋?

- 애플리케이션의 상태를 저장하고 관리하는 데 사용되는 쿠버네티스 객체
- 디플로이먼트는 파드를 삭제하고 생성할 때 상태가 유지되지 않는 한계가 있음
- 스테이트풀셋으로 생성되는 파드는 영구 식별자를 가지고 상태를 유지

- 스테이트풀셋를 사용하는 경우
  - 안정적이고 고유한 네트워크 식별자가 필요한 경우
  - 안정적이고 지속적인 스토리지
  - 질서 정연한 배치 및 확장
  - 주문, 자동 롤링 업데이트

- 스테이트풀셋의 문제
  - 스테이트풀셋과 관련된 볼륨이 삭제되지 않음
  - 파드의 스토리지는 PV나 스토리지클래스로 프로비저닝 수행해야 함
  - 롤링업데이트를 수행하는 경우 수동으로 복구해야 할 수 있음
  - 파드 네트워크 ID를 유지하기 위해 헤드레스(headless) 서비스 필요

스테이트풀셋

헤드레스 서비스 작성 요령

- 헤드레스 서비스는 clusterIP를 None으로 지정하여 생성
- 헤드레스 서비스는 IP가 할당되지 않음
- kube-proxy가 밸런싱이나 프록시 형태로 동작하지 않음

nginx-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
```

스테이트풀셋

스테이트풀셋 작성 요령

- 디플로이먼트와 전반적으로 설정 방법은 유사
- serviceName을 통해 서비스 지정
- terminationGracePeriodSeconds
  - 파드에도 들어갈 수 있는 설정으로
  - 종료 요청(SIGTERM) 후 30초 기다림
  - 컨테이너가 종료되지 않으면 강제로 SIGKILL
- volumeMounts
  - 영구 스토리지를 연결하고자 하는 위치
- (다음장에 계속)

nginx-statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.templ
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.sel
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
```

스테이트풀셋

스테이트풀셋 작성 요령

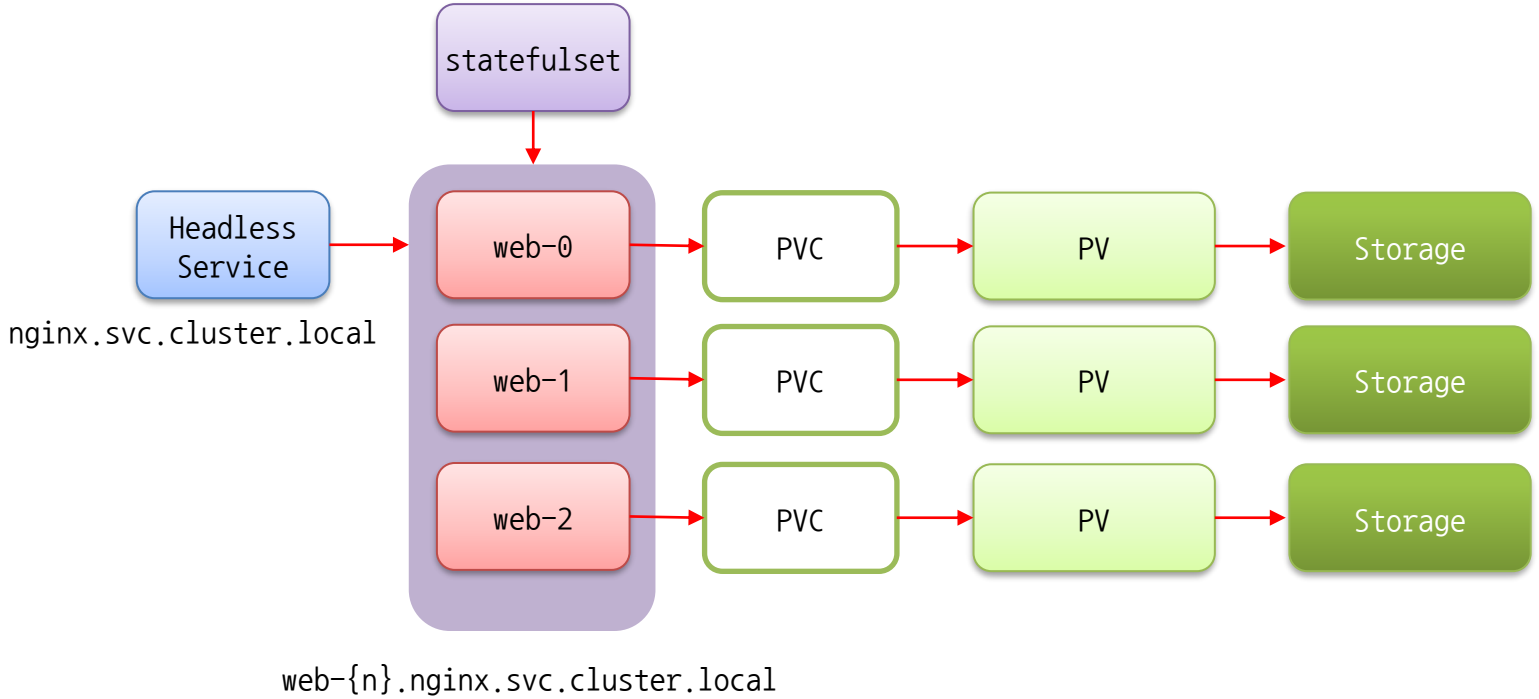
- volumeClaimTemplates을 사용해 안정적인 스토리지 제공

```
volumeClaimTemplates:
- metadata:
    name: www
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: "my-storage-class"
    resources:
      requests:
        storage: 1Gi
```

스테이트풀셋

스테이트풀셋의 다수 파드 식별 요령

- 스테이트풀셋로 다수의 파드를 생성할 수 있음
- 그러나 스테이트풀셋은 상태를 유지하는 파드로 각각의 파드를 인식할 수 있는 방법을 알아야 함
- 이를 사용해 안정적인 네트워크 ID와 스토리지를 식별할 수 있음
- 순차적으로 하나씩 배포하는데 앞의 파드가 준비상태가 돼야 다음 파드를 생성
- 배포 순서 0 -> n-1 // 종료 순서 n-1 -> 0



스테이트풀셋

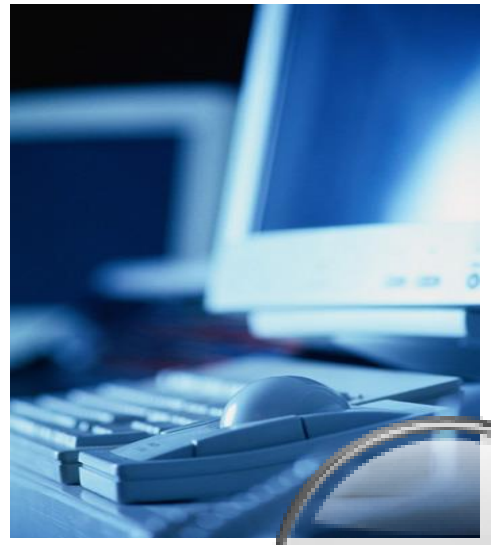
업데이트 전략

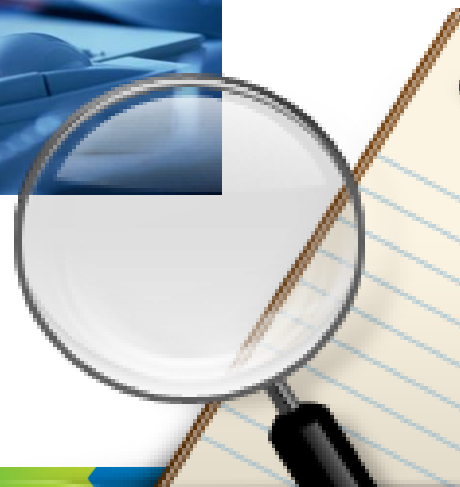
- **OnDelete**
  - 파드를 자동으로 업데이트하는 기능이 아님
  - 수동으로 삭제해주시면 스테이트풀 셋의 spec.template를 적용한 새로운 파드가 생성됨

- **RollingUpdate**
  - 롤링 업데이트를 구현하면 하번에 하나씩 파드를 업데이트
  - web-{n-1}부터 web-0의 순서로 진행

컨테이너 롤링 업데이트 전략 이해

- 디플로이먼트
- 애플리케이션 롤링 업데이트와 롤백

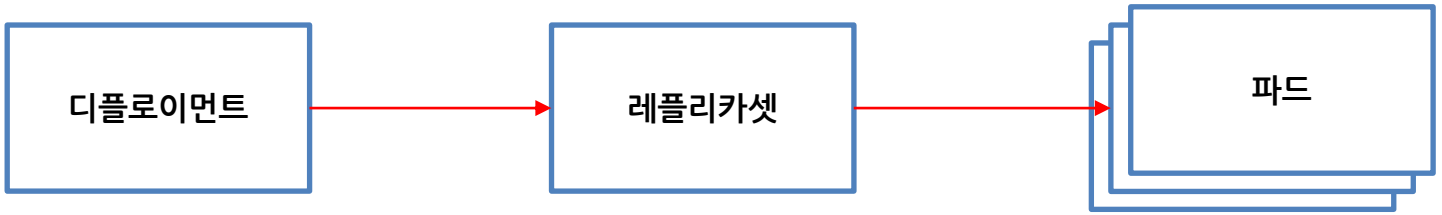




디플로이먼트

디플로이먼트

- 애플케이션을 다운 타입 없이 업데이트 가능하도록 도와주는 리소스!
- 레플리카셋과 레플리케이션컨트롤러 상위에 배포되는 리소스



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

디플로이먼트

디플로이먼트 작성 요령

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

pod.yaml

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

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

디플로이먼트

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

디플로이먼트

연습문제

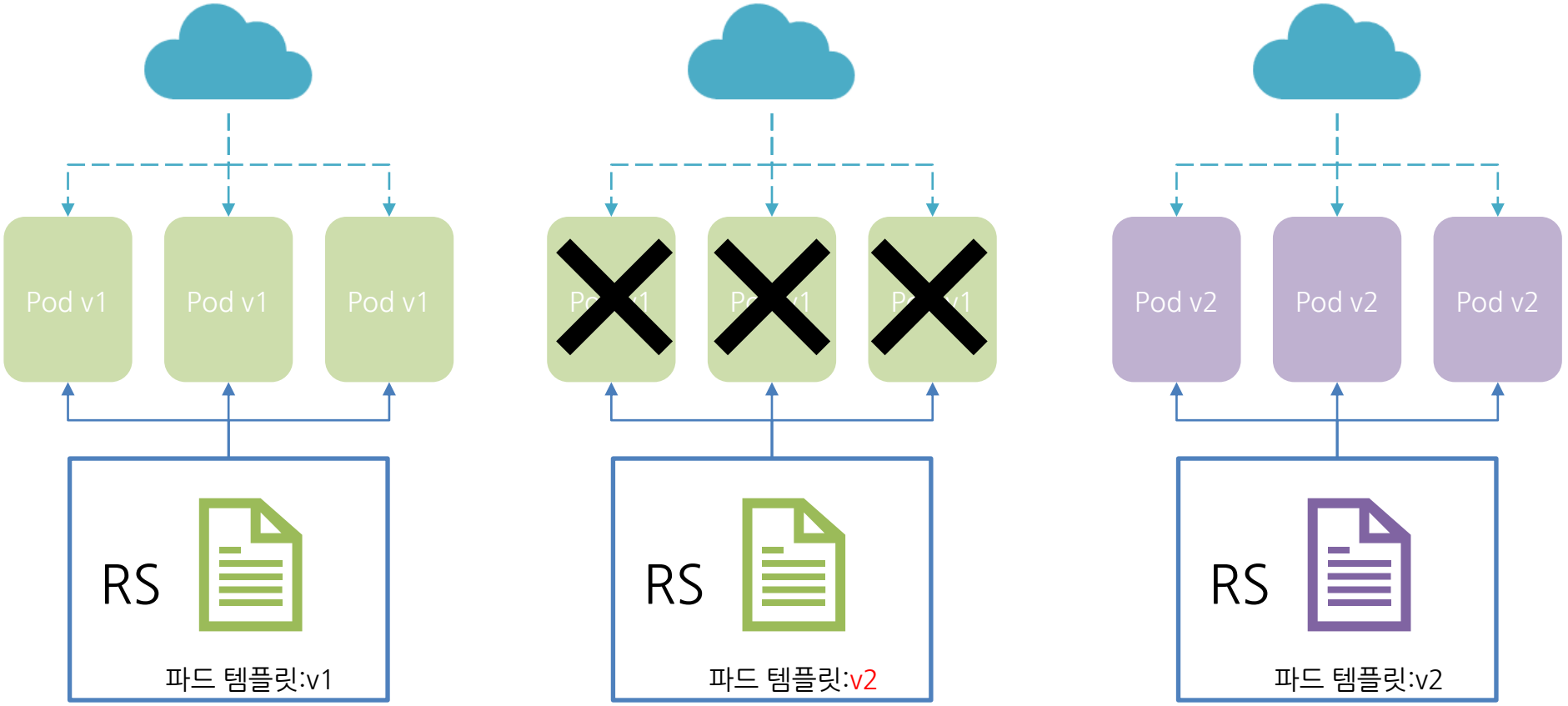
- jenkins 디플로이먼트를 deploy-jenkins를 생성하라.
- jenkins 디플로이먼트로 배포되는 앱을 app: jenkins-test로 레이블링하라.
- 디플로이먼트로 배포된 파드를 하나 삭제하고 이후 생성되는 파드를 관찰하라.
- 새로 생성된 파드의 레이블을 바꾸어 Deployment의 관리 영역에서 벗어나게 하라.
- Scale 명령을 사용해 레플리카 수를 5개로 정의한다.
- edit 기능을 사용하여 10로 스케일링하라.

애플리케이션 롤링 업데이트와 롤백

애플리케이션 롤링 업데이트와 롤백

기존 모든 파드를 삭제 후 새로운 파드 생성

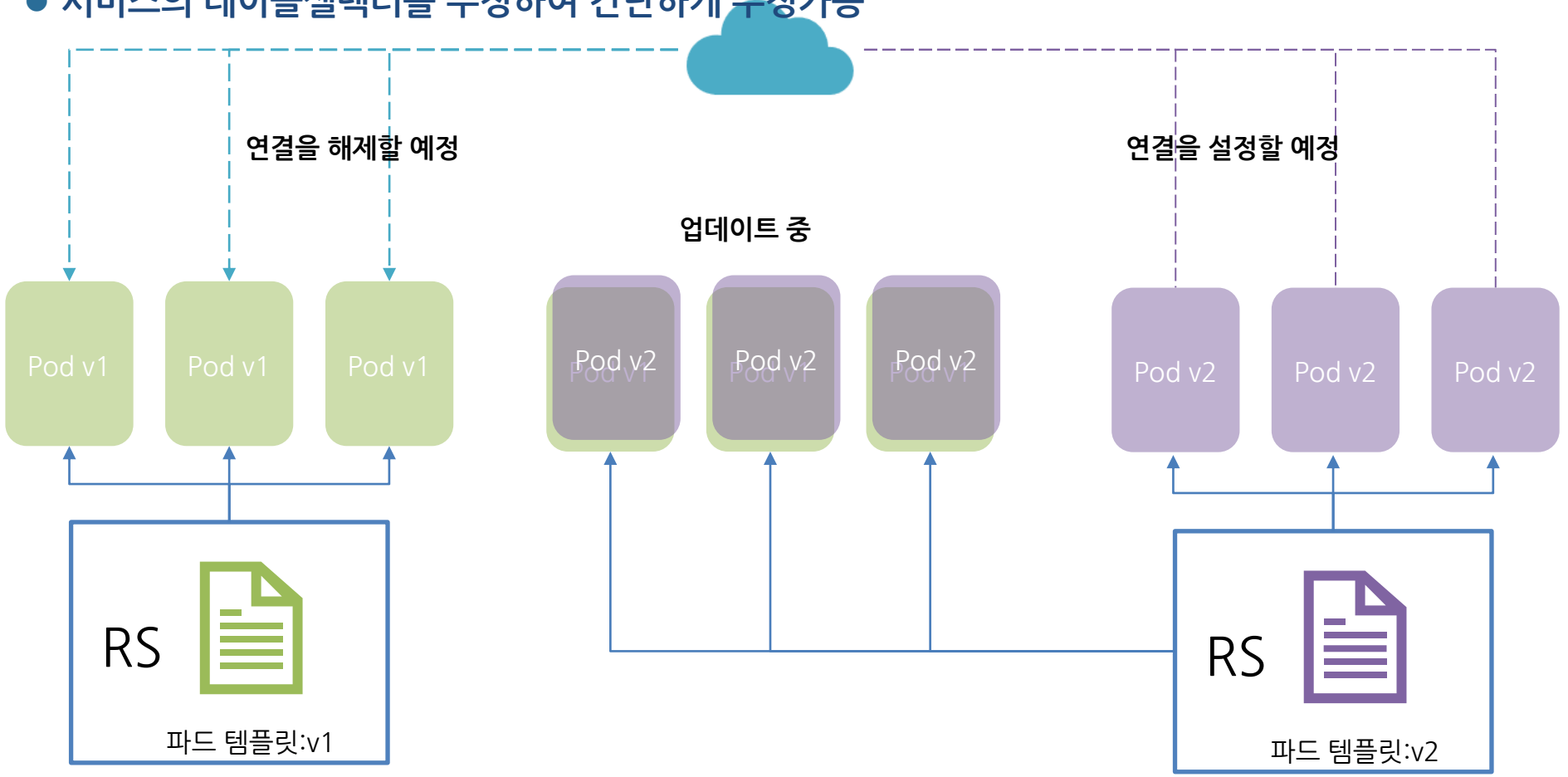
- 잠깐의 다운 타임 발생



애플리케이션 롤링 업데이트와 롤백

새로운 파드를 실행시키고 작업이 완료되면 오래된 파드를 삭제

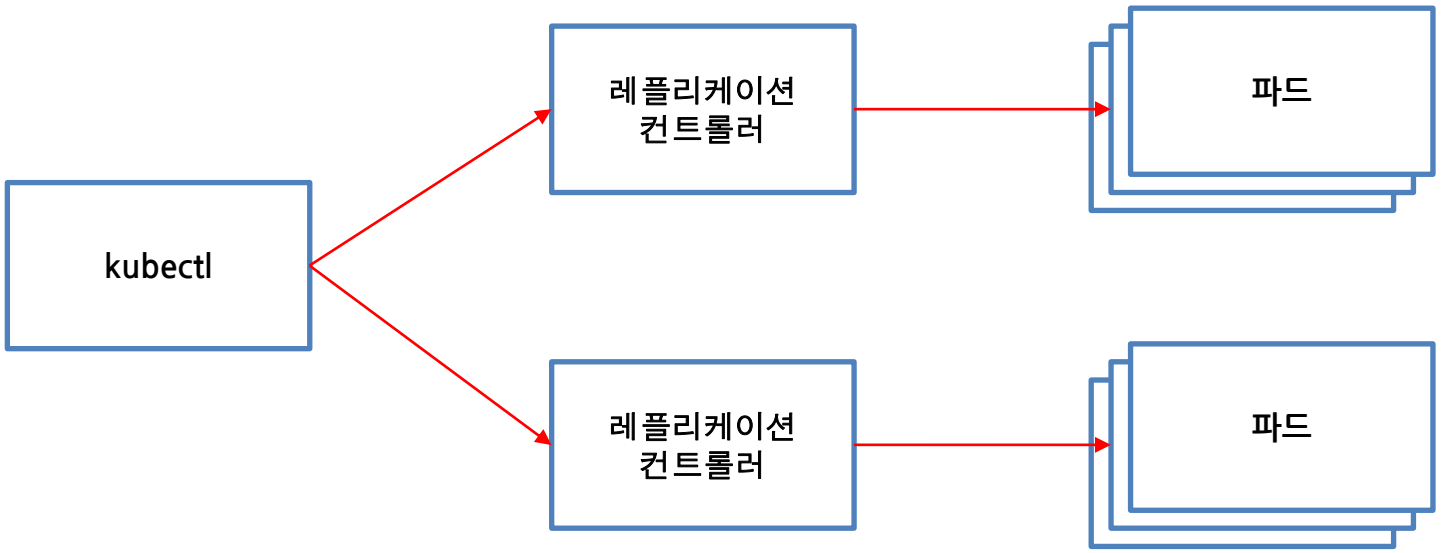
- 새 버전을 실행하는 동안 구 버전 파드와 연결
- 서비스의 레이블셀렉터를 수정하여 간단하게 수정가능



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 셀렉터, 원하는 복제본 수, 파드 템플릿
- 디플로이먼트의 전략을 yaml에 지정하여 사용 가능
- 먼저 업데이트 시나리오리를 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"]
```

main.go

```
package main

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

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

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

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

- **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 모니터하는 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

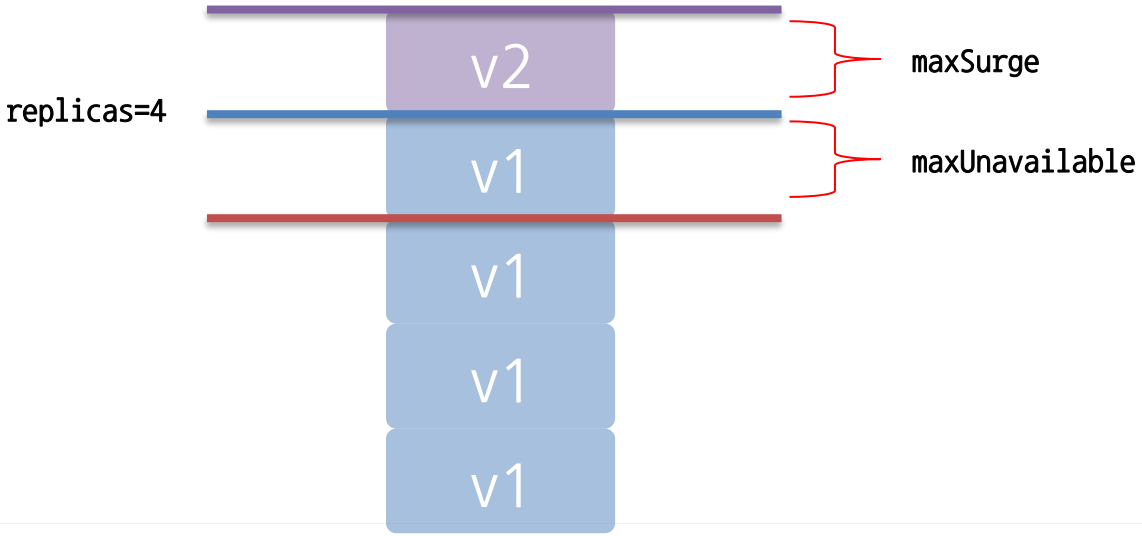
```
$ kubectl set image deployment http-go http-go=gasbugs/http-go:v3
deployment.extensions/http-go image updated

$ 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 파드 운영)

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




```yaml
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnvailable: 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 mongo 이미지를 사용하여 업데이트와 롤백을 실행하라.**
  - 모든 revision 내용은 기록돼야 한다.
  - mongo:4.2 이미지를 사용하여 deployment를 생성하라.
    - ✓ Replicas: 10
    - ✓ maxSurge: 50%
    - ✓ maxUnavailable: 50%
  - mongo:4.4 롤링 업데이트를 수행하라.
  - mongo:4.2로 롤백을 수행하라.

차트 패키징 및 github 레파지토리를 활용한 배포

헬름 레파지토리 파일 깃헙에 업로드하기

- index.yaml 파일 위치를 github에서 찾아서 경로를 add하고 업데이트

```
helm repo add gasbugs-charts https://raw.githubusercontent.com/gasbugs/helm-charts/mai
helm repo update
```

- serach 명령을 사용해 업로드한 두 차트를 검색

```
$ helm search repo mychart
NAME                        CHART VERSION   APP VERSION     DESCRIPTION
gasbugs-charts/mychart      0.1.0           1.16.0          A Helm chart for Kubernetes
gasbugs-charts/mychart2     0.1.0           1.16.0          A Helm chart for Kubernetes
```

- 헬름 차트를 사용해 인스톨 시작

```
helm install mychart-test gasbugs-charts/mychart2 -n default
```