Managing Kubernetes

2021-05-08
written by whatwant

Agenda

Chapter1. Kubernetes Overview

1주차: **Docker and Kubernetes**

Chapter2. Kubernetes Core

2주차: **Environment & POD**

3주차: **Replication and other controllers**

4주차: **Services**

5주차: **Volumes (Service 보충 수업 포함)**

6주차: **ConfigMaps and Secrets & Kubernetes REST API**

7주차: **Deployment & StatefulSet**

Chapter3. Kubernetes Managing

8주차: **Authentication and User Management & Authorization & Admission Control**

9주차: **Networking**

10주차: **Monitoring**

11주차: **Disaster Recovery**

※ 참고 : https://home.modulabs.co.kr/product/managing-kubernetes/

5 week
Volumes (+Service)

Supplementary Lessons

DNS – Trouble Shooting - 1/2

- `/etc/resolv.conf`에 원하지 않는 값이 반영되어, 잘못된 결과가 나오는 경우

```
❯ kubectl exec -it rs-node-web-j7g25 -- cat /etc/resolv.conf
nameserver 169.254.25.10
search default.svc.cluster.local svc.cluster.local cluster.local skbroadband
options ndots:5

❯ kubectl exec -it rs-node-web-j7g25 -- curl -s http://svc-node.default.svc.cluster.local
command terminated with exit code 7
```

Node의 `/etc/resolv.conf`에 있는 정보를
K8s의 coredns가 읽어와서
Pod의 Container 내부의
`/etc/resolv.conf`를 생성하기에
왼쪽과 같은 사태가 발생된다.

- DHCP 사용하는 경우, DNS 정보가 `/etc/resolv.conf`에 반영되어서 발생
  → Static IP 설정을 하면 해결됨

```
❯ sudo nano /etc/netplan/00-installer-config.yaml

network:
  ethernets:
    enp0s3:
      dhcp4: no
      dhcp6: no
      addresses: [192.168.100.111/24]
      gateway4: 192.168.100.1
      nameservers:
        addresses: [8.8.8.8,8.8.4.4]
  version: 2
```

각자 환경에 맞춰서
모든 Node를 고정 IP 설정하면 된다.

DNS – Trouble Shooting – 2/2

- Kubernetes는 각 container의 /etc/resolv.conf 파일을 수정
- FQDN (Fully Qualified Domain Name) : service name . namespace name . svc.cluster.local

```
❯ kubectl create –f rs-node-web.yaml
❯ kubectl create –f svc-none-node-web.yaml

❯ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
rs-node-web-8jm4f   1/1     Running   1          19h
rs-node-web-cjr6x   1/1     Running   1          19h
rs-node-web-hpxfh   1/1     Running   1          19h

❯ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.233.0.1      <none>        443/TCP   21h
svc-node     ClusterIP   10.233.61.192   <none>        80/TCP    19h

❯ kubectl exec -it rs-node-web-8jm4f -- cat /etc/resolv.conf
nameserver 169.254.25.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

```
❯ kubectl exec -it rs-node-web-8jm4f -- curl -s http://svc-node
You've hit rs-node-web-cjr6x

❯ kubectl exec -it rs-node-web-8jm4f -- curl -s http://svc-node.default
You've hit rs-node-web-hpxfh

❯ kubectl exec -it rs-node-web-8jm4f -- curl -s http://svc-node.default.svc
You've hit rs-node-web-8jm4f

❯ kubectl exec -it rs-node-web-8jm4f -- curl -s http://svc-node.default.svc.cluster
command terminated with exit code 6

❯ kubectl exec -it rs-node-web-8jm4f -- curl -s http://svc-node.default.svc.cluster local
You've hit rs-node-web-cjr6x
```

ExternalName

- 일반적인 selector에 대한 Service가 아닌, DNS 이름에 대한 Service에 매핑
- ExternalName은 IPv4 주소 문자열 및 숫자로 구성된 DNS 이름도 허용
  . But, IPv4 주소와 유사한 ExternalName은 CoreDNS 또는 ingress-nginx에 의해 확인되지 않음
  . ExternalName은 정식(canonical) DNS 이름을 지정하기 때문 (Proxy, Forwarding이 아닌 DNS라는 점이 중요!!!)
  . IP 주소를 하드 코딩하려면, 헤드리스(headless) 서비스 사용 권장
- HTTP 및 HTTPS를 포함한, 몇몇 일반적인 프로토콜에 ExternalName을 사용하는 것은 문제점 존재
  . 클러스터 내부의 클라이언트가 사용하는 호스트 이름(hostname)이 ExternalName이 참조하는 이름과 다름
  . 호스트 이름을 사용하는 프로토콜의 경우, 이러한 차이로 인해 오류가 발생하거나 예기치 않은 응답이 발생할 수 있음
  . HTTP 요청에는 오리진(origin) 서버가 인식하지 못하는 `Host :` 헤더가 존재
  . TLS 서버는 클라이언트가 연결된 호스트 이름과 일치하는 인증서를 제공할 수 없음

※ 참고 : https://kubernetes.io/ko/docs/concepts/services-networking/service/#externalname

Ingress 실습 - 1/2

```
❯ kubectl --namespace ingress-nginx get pods -o wide
NAME                                 READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
ingress-nginx-controller-4cf6w   1/1     Running   1          23h   10.233.110.8    worker1   <none>           <none>
ingress-nginx-controller-lfbhm   1/1     Running   1          23h   10.233.103.5    worker2   <none>           <none>
```

- 위 예제는 Kubespray를 이용하여 K8s를 설치하면서, ingress 부분을 true 설정을 해서 설치된 경우이다.
- controller가 구동되는 NODE를 잘 살펴보자.

```
❯ kubectl get services –o wide
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE    SELECTOR
kubernetes       ClusterIP      10.233.0.1      <none>        443/TCP        23h    <none>
svc-node         ClusterIP      10.233.61.192   <none>        80/TCP         21h    app=node-web
```

- Ingress 설정하기에 앞서서, Ingress에서 연결할 Service를 미리 등록해 놓아야 한다.
- 일단은 selector를 통해 Pod를 관리하는 Service로 만들어봤다.

```
❯ kubectl create –f svc-ingress-service.yaml
```

- yaml 파일 내용을 잘 살펴보기 바란다.

```
❯ kubectl get ingresses -o wide
NAME          CLASS    HOSTS              ADDRESS                          PORTS   AGE
svc-ingress   <none>   k8s.whatwant.com   192.168.100.112,192.168.100.113   80      107m
```

- HOSTS 내용과 ADDRESS 정보가 중요하다.
- ADDRESS의 IP와 HOSTS가 mapping되면 된다.

svc-none-node-web.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-node

spec:
  ports:
  - port: 80
    targetPort: 8080

  selector:
    app: node-web
```

svc-ingress-service.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: svc-ingress

spec:
  rules:
  - host: k8s.whatwant.com
    http:
      paths:
      - pathType: Prefix
        path: /test
        backend:
          service:
            name: svc-node
            port:
              number: 80
```

Ingress 실습 - 2/2

- Ingress 정보에서 보이는 HOSTS와 ADDRESS에 맞춰서 접근할 수 있도록 해줘야 한다.

  . 실제 Domain 설정이 되어 있다고 하면 이를 이용하면 되지만,

  . 임의의 HOSTS로 설정했다면, 접근하는 PC의 hosts 정보에 임의의 정보를 적어줘야 한다.

```
[Windows] C:\Windows\System32\drivers\etc\hosts
[Linux] /etc/hosts
```

```
192.168.100.112 k8s.whatwant.com
```

- `path` 설정에 따라 접근하면 잘 나온다 !!!

  . 그냥 `/` 경로로 접근하면 404 결과가 나온다

readinessProbe – 1/2

- livenessProbe와 방식은 거의 동일

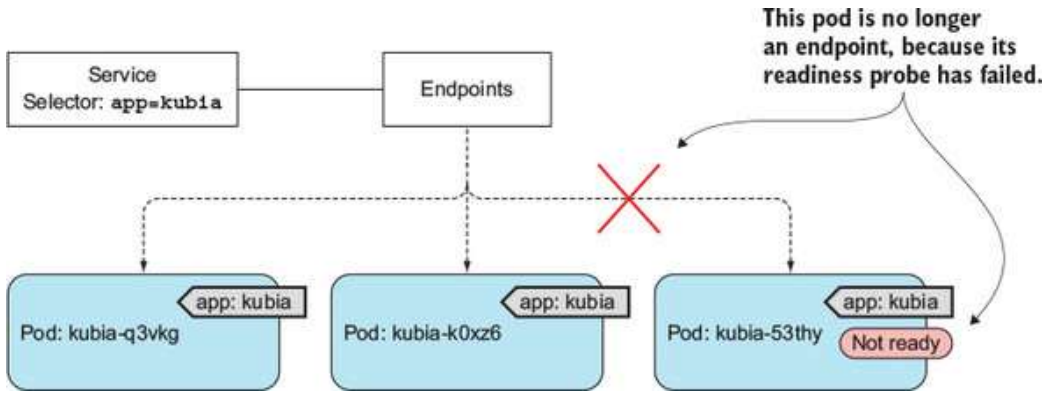
  . **HTTP GET Probe**: HTTP GET 요청의 응답 코드를 확인 (2xx, 3xx이면 성공)

  . **TCP Socket Probe**: TCP 연결 성공 여부

  . **Exec Probe**: 명령을 실행하고 exit code가 0이면 성공

- Probe가 실패하면 Service에서 제거, 성공하면 다시 추가

  . 실패한다고 해서 Container 종료/재시작 하지 않음



※ 참고 : https://livebook.manning.com/book/kubernetes-in-action/chapter-5/333

readinessProbe – 2/2

rs-node-web-readiness.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: rs-node-web

spec:
  replicas: 3

  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - node-web

  template:
    metadata:
      labels:
        app: node-web

    spec:
      containers:
      - name: node-web
        image: whatwant/node-web:1.0
        ports:
        - containerPort: 8080

        readinessProbe:
          exec:
            command:
            - ls
            - /var/ready
```

※ readinessProbe 정의는 Pod에서 이루어진다 !!!

```
❯ ❯ kubectl apply -f rs-node-web-readiness.yaml

❯ ❯ kubectl delete pods --selector=app=node-web

❯ kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
rs-node-web-hchct   0/1     Running   0          79s   10.233.110.10   worker1   <none>           <none>
rs-node-web-qsmvn   0/1     Running   0          79s   10.233.110.9    worker1   <none>           <none>
rs-node-web-wt49r   0/1     Running   0          79s   10.233.103.9    worker2   <none>           <none>

❯ kubectl exec -it rs-node-web-hchct -- touch /var/ready

❯ kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE     IP              NODE      NOMINATED NODE   READINESS GATES
rs-node-web-hchct   1/1     Running   0          3m30s   10.233.110.10   worker1   <none>           <none>
rs-node-web-qsmvn   0/1     Running   0          3m30s   10.233.110.9    worker1   <none>           <none>
rs-node-web-wt49r   0/1     Running   0          3m30s   10.233.103.9    worker2   <none>           <none>
```

Headless Service - 1/2

- Why?

  . 임의의 Pod 하나에 연결하는 것이 아니라, 생성된 모든 Pod에 연결하고 싶은 경우

  . Pod 끼리 통신이 필요한 경우

- How

  . DNS 조회를 통해 IP 반환 (Pod IP 전체를...)

svc-headless-node-web.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-node-headless

spec:
  clusterIP: None

  ports:
  - port: 80
    targetPort: 8080

  selector:
    app: node-web
```

```
❯ kubectl create -f svc-headless-node-web.yaml
service/svc-node-headless created

❯ kubectl get services -o wide
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE    SELECTOR
kubernetes         ClusterIP   10.233.0.1      <none>        443/TCP        27h    <none>
svc-node           ClusterIP   10.233.61.192   <none>        80/TCP         26h    app=node-web
svc-node-headless  ClusterIP   None            <none>        80/TCP         5s     app=node-web

❯ kubectl apply -f https://k8s.io/examples/admin/dns/dnsutils.yaml
pod/dnsutils created
```

※ nslookup 명령어를 지원하는 Pod를 얻기 위해서...

※ 참고 : https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/

Headless Service - 2/2

```
❯ kubectl exec -it dnsutils -- nslookup svc-node-headless
Server:         169.254.25.10
Address:        169.254.25.10#53

Name:   svc-node-headless.default.svc.cluster.local
Address: 10.233.110.10
Name:   svc-node-headless.default.svc.cluster.local
Address: 10.233.110.9
Name:   svc-node-headless.default.svc.cluster.local
Address: 10.233.103.9
```

※ 등록되어 있는 모든 Pod의 IP를 확인할 수 있다!

```
❯ kubectl exec -it dnsutils -- nslookup svc-node
Server:         169.254.25.10
Address:        169.254.25.10#53

Name:   svc-node.default.svc.cluster.local
Address: 10.233.61.192
```

※ headless가 아닌 Service를 조회하면 하나만 나온다.

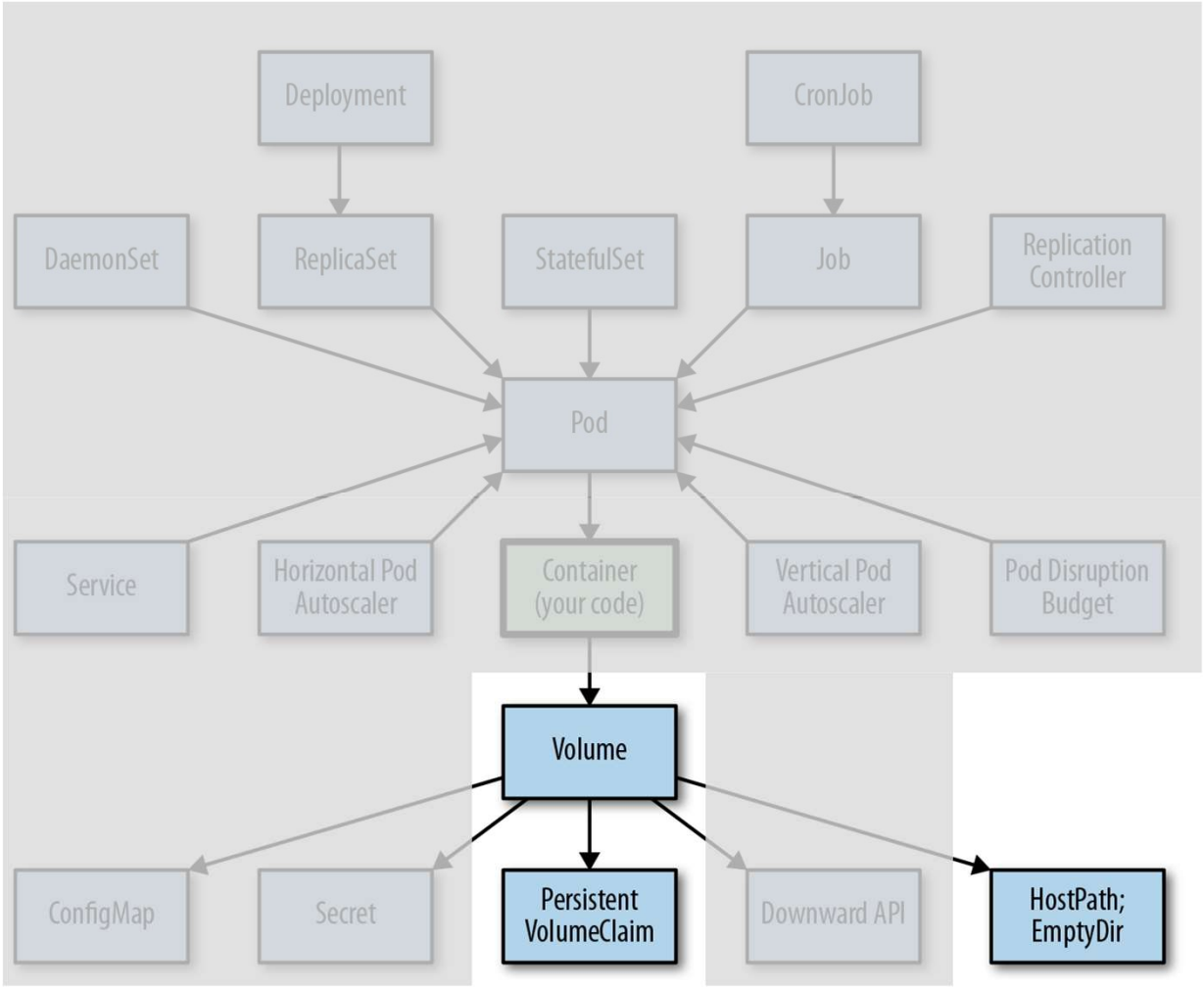
Kubernetes Network
심화학습

https://www.slideshare.net/InfraEngineer/meetup2nd-v12

https://kahoot.it/

Today ...

Volume



Volume

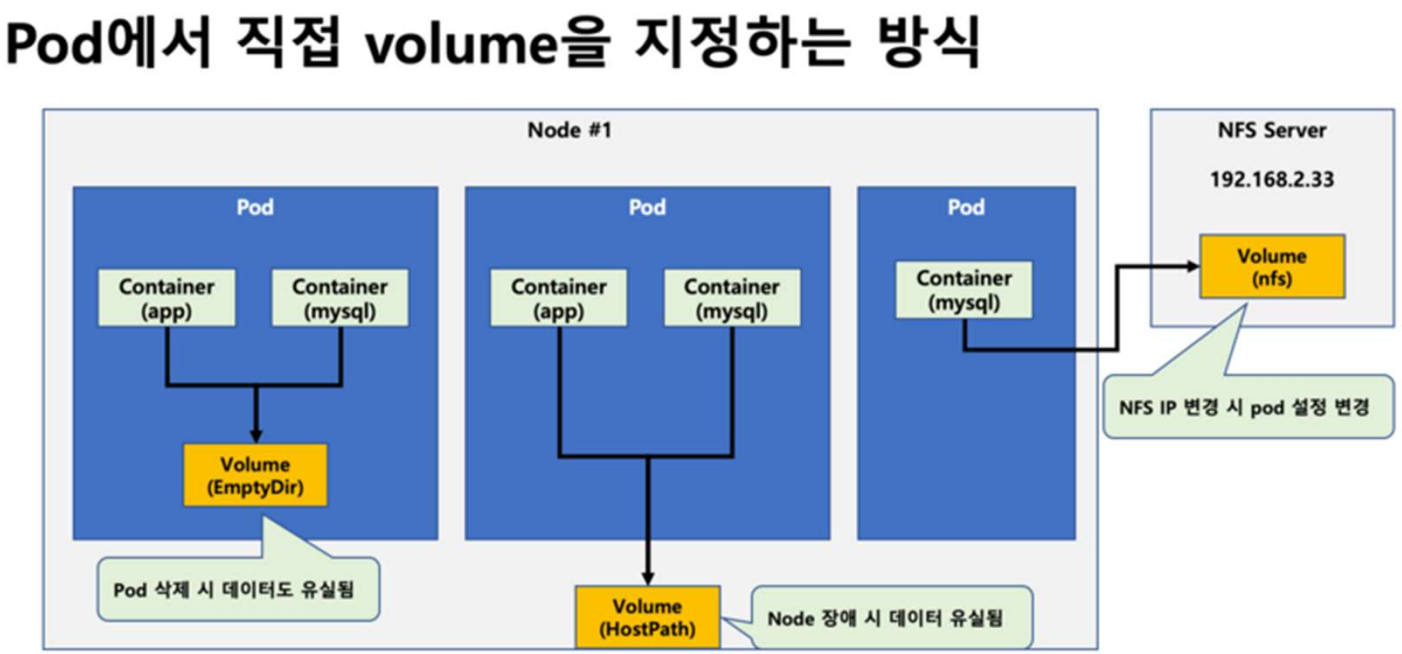
kinds of volume

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- csi
- downwardAPI
- emptyDir
- fc (파이버 채널)
- flexVolume
- flocker
- gcePersistentDisk
- gitRepo (사용중단(deprecated))
- glusterfs
- hostPath
- iscsi
- local
- nfs
- persistentVolumeClaim
- projected
- portworxVolume
- quobyte
- rbd
- scaleIO
- secret
- storageos
- vsphereVolume

"Kubernetes In Action" 책에서 설명하는 범위

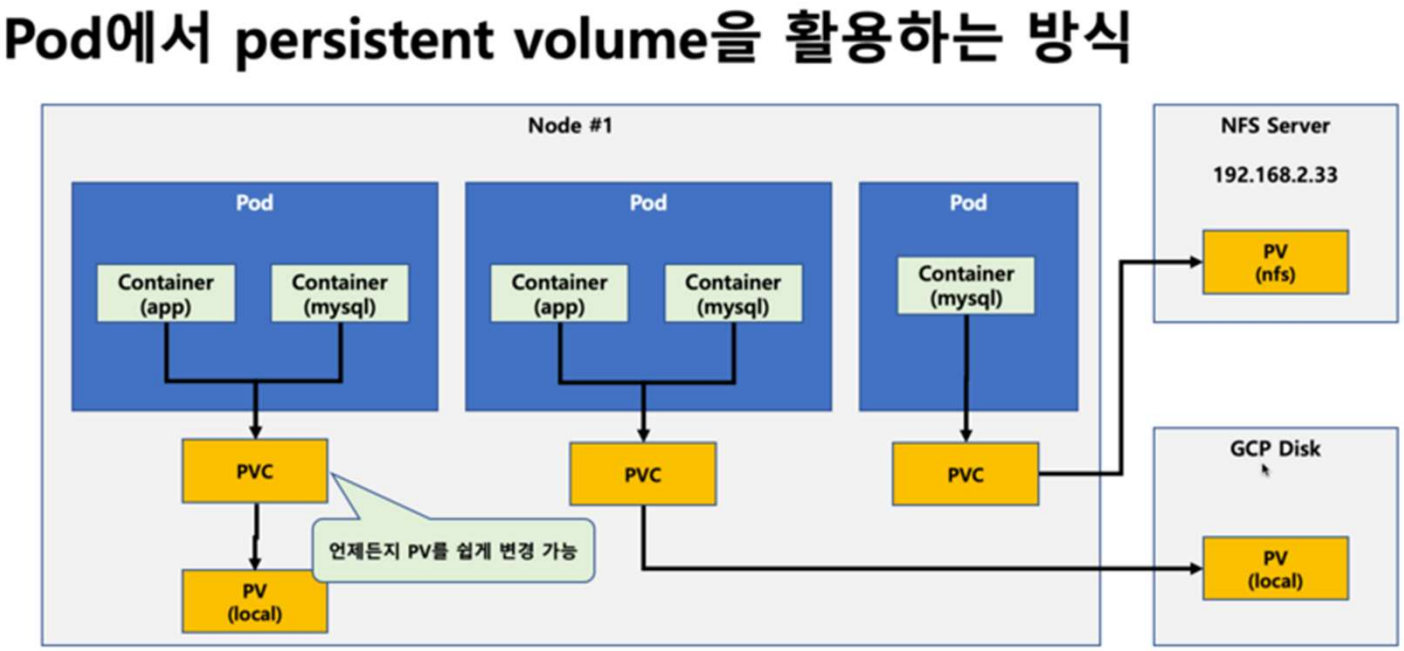
※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/volumes/

Volume – 1/2



※ 참고 : https://m.blog.naver.com/freepsw/222005161870

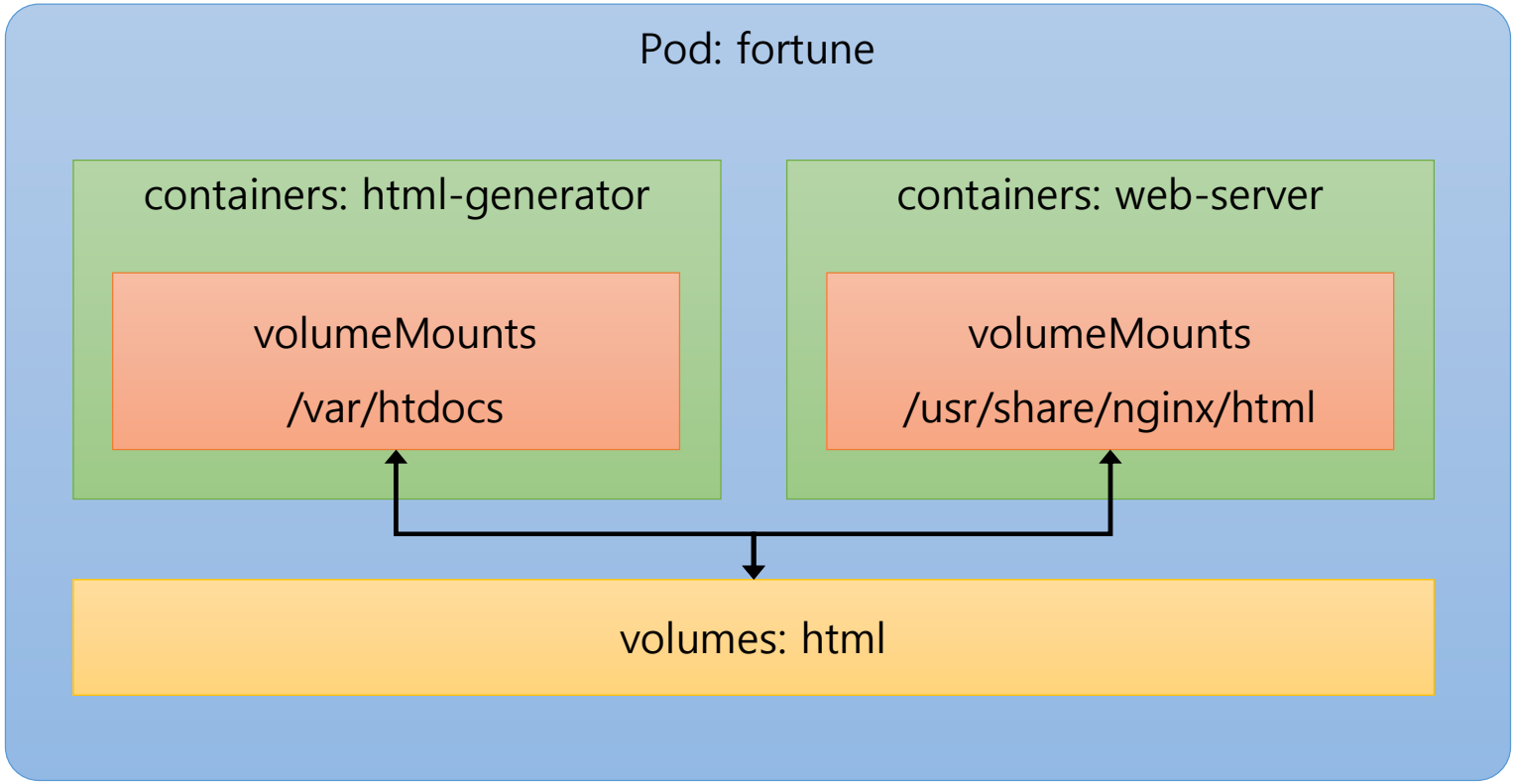
Volume – 2/2



※ 참고 : https://m.blog.naver.com/freepsw/222005161870

emptyDir – 1/3

- Pod 안에서 Container끼리 공유 또는 Container가 재시작 하더라도 저장된 파일 유지
- Pod가 재시작 하는 경우에는 저장된 파일 유실



emptyDir – 2/3

emptyDir.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: emptydir

spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu

  template:
    metadata:
      labels:
        app: ubuntu

    spec:
      containers:
      - image: ubuntu:20.04
        name: worker1
        command: ["/bin/sleep", "3650d"]
        volumeMounts:
        - name: emptydir-demo
          mountPath: /data/emptydir1

      - image: ubuntu:20.04
        name: worker2
        command: ["/bin/sleep", "3650d"]
        volumeMounts:
        - name: emptydir-demo
          mountPath: /data/emptydir2

      volumes:
      - name: emptydir-demo
        emptyDir: {}
```

```
❯ kubectl exec -it emptydir-qjq5w -c worker1 -- ls -al /data
total 12
drwxr-xr-x 3 root root 4096 May  7 17:17 .
drwxr-xr-x 1 root root 4096 May  7 17:17 ..
drwxrwxrwx 2 root root 4096 May  7 17:17 emptydir1

❯ kubectl exec -it emptydir-qjq5w -c worker2 -- ls -al /data
total 12
drwxr-xr-x 3 root root 4096 May  7 17:17 .
drwxr-xr-x 1 root root 4096 May  7 17:17 ..
drwxrwxrwx 2 root root 4096 May  7 17:17 emptydir2
```

※ 다른 이름으로 mount 가능

```
❯ kubectl exec -it emptydir-qjq5w -c worker1 -- touch /data/emptydir1/wow

❯ kubectl exec -it emptydir-qjq5w -c worker1 -- ls -al /data/emptydir1/
total 8
drwxrwxrwx 2 root root 4096 May  7 17:20 .
drwxr-xr-x 3 root root 4096 May  7 17:17 ..
-rw-r--r-- 1 root root    0 May  7 17:20 wow

❯ kubectl exec -it emptydir-qjq5w-qjq5w -c worker2 -- ls -al /data/emptydir2/
total 8
drwxrwxrwx 2 root root 4096 May  7 17:20 .
drwxr-xr-x 3 root root 4096 May  7 17:17 ..
-rw-r--r-- 1 root root    0 May  7 17:20 wow
```

※ 같은 공간을 공유하고 있음을 확인

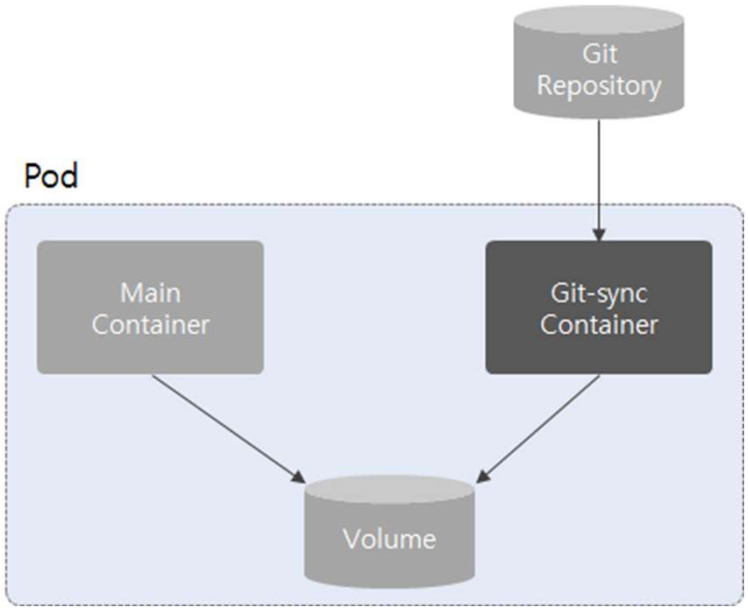
emptyDir – 3/3

```
❯ kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE     IP              NODE      NOMINATED NODE   READINESS GATES
emptydir-qjq5w   2/2     Running   0          2m27s   10.233.103.38   worker2   <none>           <none>

❯ kubectl delete pods emptydir-qjq5w
pod "emptydir-qjq5w" deleted

❯ kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
emptydir-z5xzv   2/2     Running   0          57s   10.233.110.24   worker1   <none>           <none>

❯ kubectl exec -it emptydir-z5xzv -c worker1 -- ls -al /data/emptydir1/
total 8
drwxrwxrwx 2 root root 4096 May  7 17:31 .
drwxr-xr-x 3 root root 4096 May  7 17:31 ..

❯ kubectl exec -it emptydir-z5xzv -c worker2 -- touch /data/emptydir2/wow

❯ kubectl exec -it emptydir-z5xzv -c worker1 -- ls -al /data/emptydir1/
total 8
drwxrwxrwx 2 root root 4096 May  7 17:33 .
drwxr-xr-x 3 root root 4096 May  7 17:31 ..
-rw-r--r-- 1 root root    0 May  7 17:33 wow
```

※ Pod가 재시작 되면 저장된 내용이 사라짐 !!!

gitRepo : Deprecated → ‘git-sync’ Sidecar Container

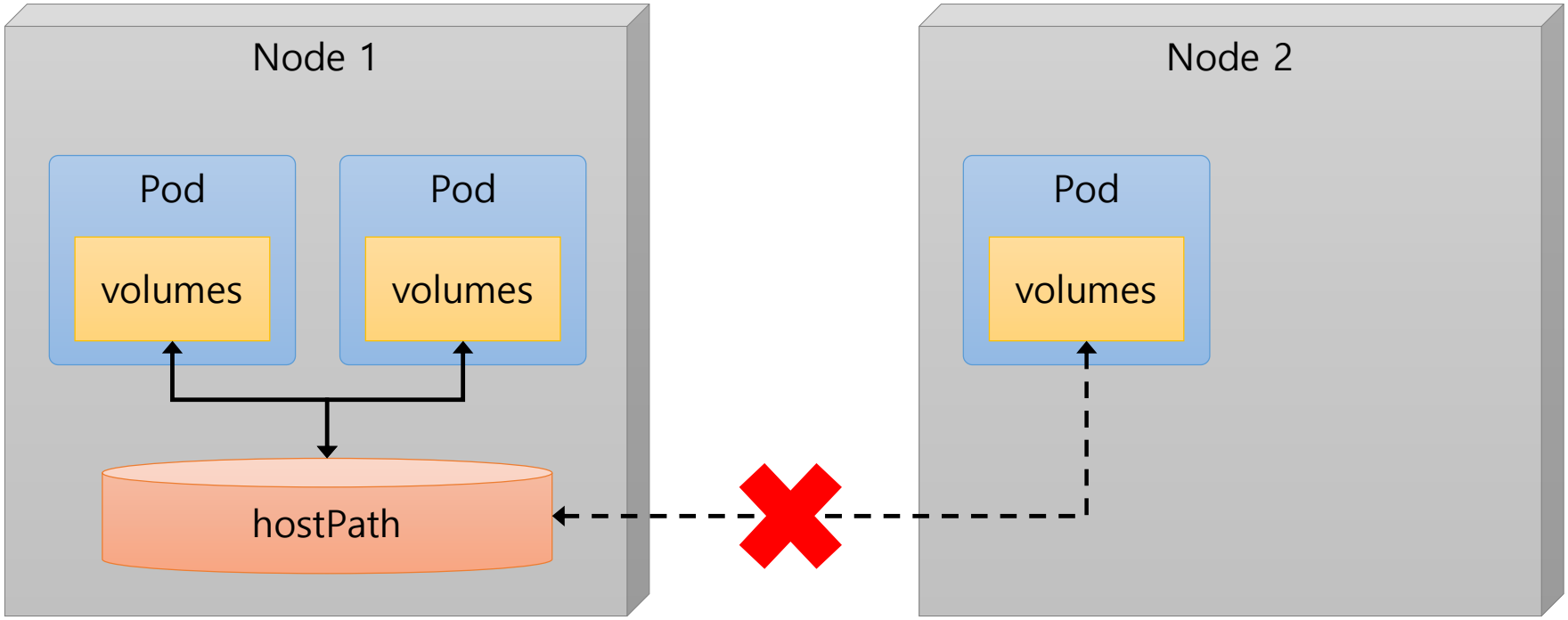
- gitRepo 대신 git-sync의 sidecar container를 구성하는 것으로 검토하면 좋다.



※ 참고 : https://github.com/kubernetes/git-sync

hostPath – 1/4

- hostPath는 다른 Node에서는 참조할 수 없다.



hostPath – 2/4

- hostPath 볼륨에 대한 type

값	행동
	빈 문자열 (기본값)은 이전 버전과의 호환성을 위한 것으로, hostPath 볼륨은 마운트 하기 전에 아무런 검사도 수행되지 않는다.
DirectoryOrCreate	만약 주어진 경로에 아무것도 없다면, 필요에 따라 Kubelet이 가지고 있는 동일한 그룹과 소유권, 권한을 0755로 설정한 빈 디렉터리를 생성한다.
Directory	주어진 경로에 디렉터리가 있어야 함
FileOrCreate	만약 주어진 경로에 아무것도 없다면, 필요에 따라 Kubelet이 가지고 있는 동일한 그룹과 소유권, 권한을 0644로 설정한 빈 디렉터리를 생성한다.
File	주어진 경로에 파일이 있어야 함
Socket	주어진 경로에 UNIX 소캣이 있어야 함
CharDevice	주어진 경로에 문자 디바이스가 있어야 함
BlockDevice	주어진 경로에 블록 디바이스가 있어야 함

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/volumes/

hostPath – 3/4

hostPath.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: hostpath

spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu

  template:
    metadata:
      labels:
        app: ubuntu

    spec:
      containers:
      - image: ubuntu:20.04
        name: worker
        command: ["/bin/sleep", "3650d"]
        volumeMounts:
        - name: hostpath-demo
          mountPath: /data/hostpath

      volumes:
      - name: hostpath-demo
        hostPath:
          path: /tmp/hostpath-demo
          type: DirectoryOrCreate
```

```
❯ kubectl apply -f hostPath.yaml
replicaset.apps/hostpath created

❯ kubectl exec -it hostpath-5z5fh -- ls -al /data/
total 12
drwxr-xr-x 3 root root 4096 May  7 18:13 .
drwxr-xr-x 1 root root 4096 May  7 18:13 ..
drwxr-xr-x 2 root root 4096 May  7 18:13 hostpath

❯ kubectl exec -it hostpath-5z5fh -- ls -al /data/hostpath
total 8
drwxr-xr-x 2 root root 4096 May  7 18:13 .
drwxr-xr-x 3 root root 4096 May  7 18:13 ..

❯ kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE    IP              NODE      NOMINATED NODE   READINESS GATES
hostpath-5z5fh   1/1     Running   0          103s   10.233.103.39   worker2   <none>           <none>
```

```
❯ ssh whatwant@worker2

❯ ls -al /tmp/hostpath-demo
total 8
drwxr-xr-x  2 root root 4096  5월  8 03:13 .
drwxrwxrwt 11 root root 4096  5월  8 03:15 ..
```

※ Pod가 생성된 Node에 접속해서 경로 확인

hostPath – 4/4

```
› sudo touch /tmp/hostpath-demo/wow
```

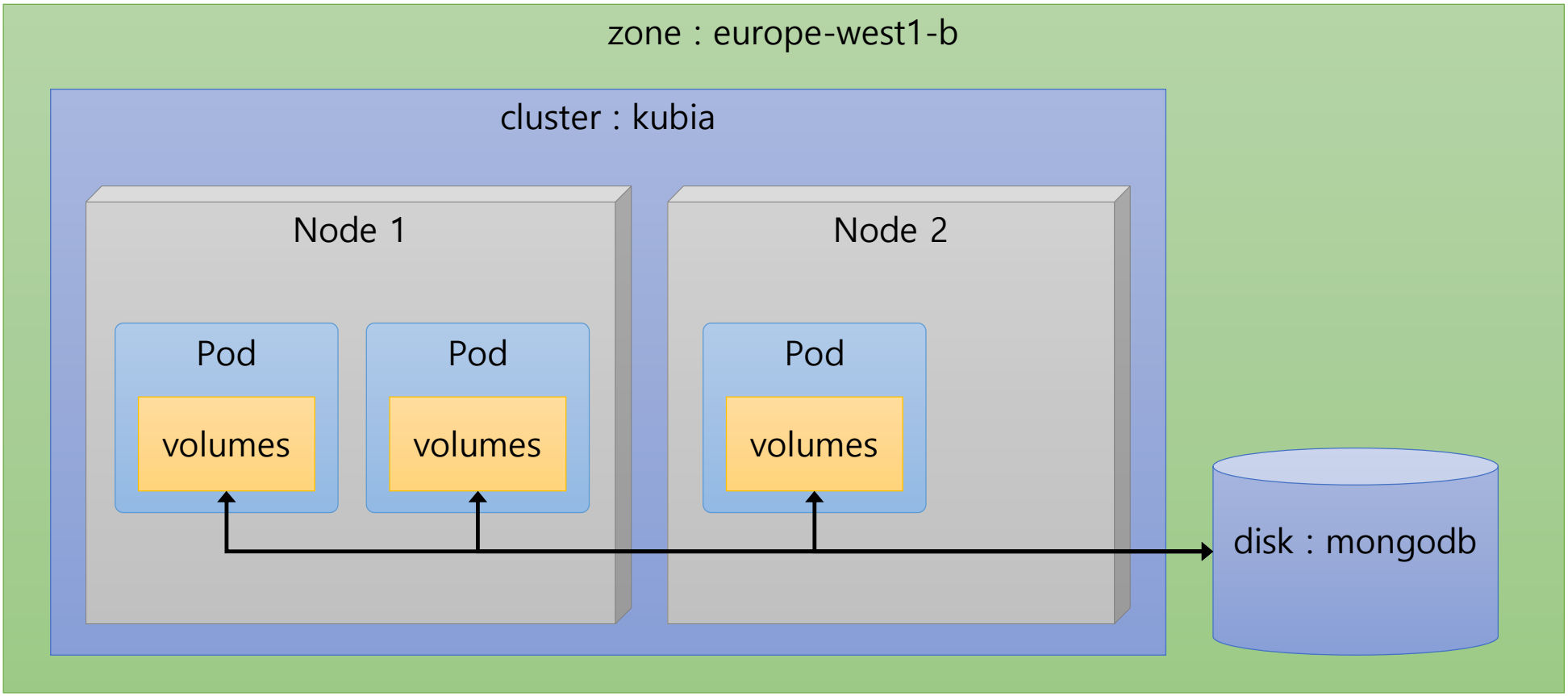
```
› kubectl exec -it hostpath-5z5fh -- ls -al /data/hostpath
total 8
drwxr-xr-x 2 root root 4096 May  7 18:20 .
drwxr-xr-x 3 root root 4096 May  7 18:13 ..
-rw-r--r-- 1 root root    0 May  7 18:20 wow

› kubectl exec -it hostpath-5z5fh -- touch /data/hostpath/wow2
```

```
› ls -al /tmp/hostpath-demo
total 8
drwxr-xr-x  2 root root 4096  5월  8 03:21 .
drwxrwxrwt 11 root root 4096  5월  8 03:21 ..
-rw-r--r--  1 root root    0  5월  8 03:20 wow
-rw-r--r--  1 root root    0  5월  8 03:21 wow2
```

※ Node와 Pod(Container) 間 파일 공유 가능

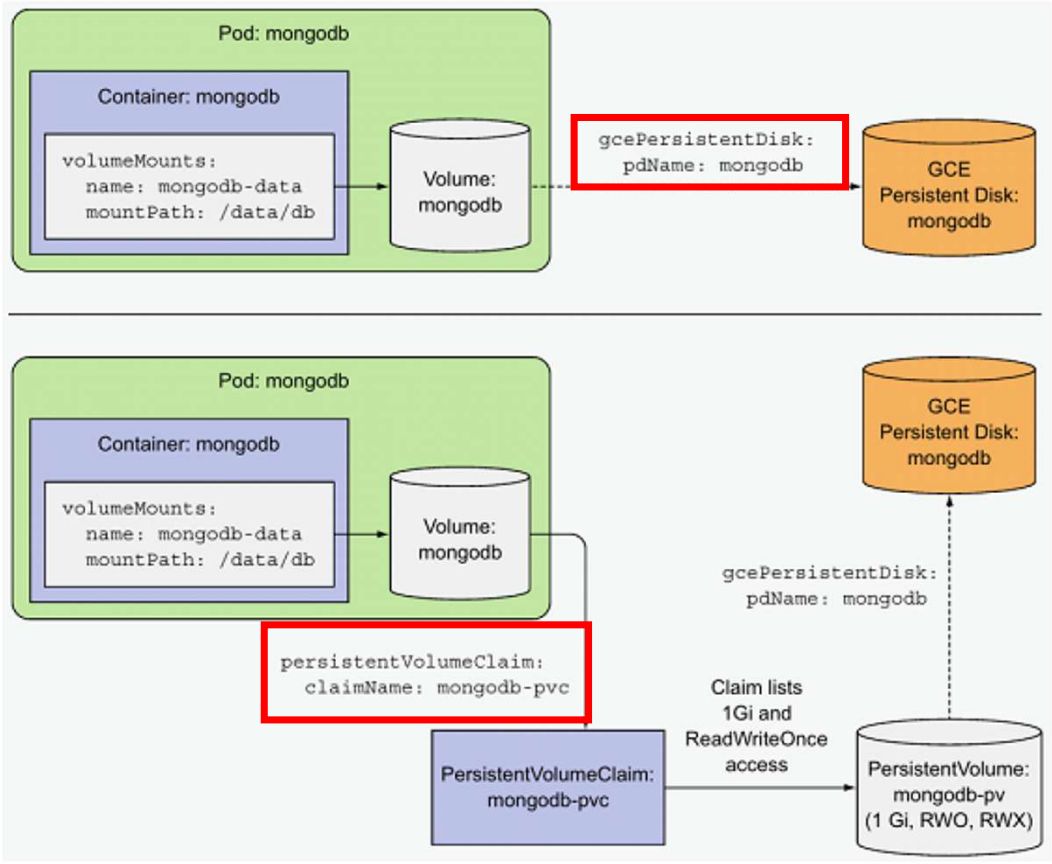
Persistent Disk (GCE)

- Public Cloud에서 제공해주는 Disk 사용



‘Persistent Disk’ vs ‘PV & PVC’

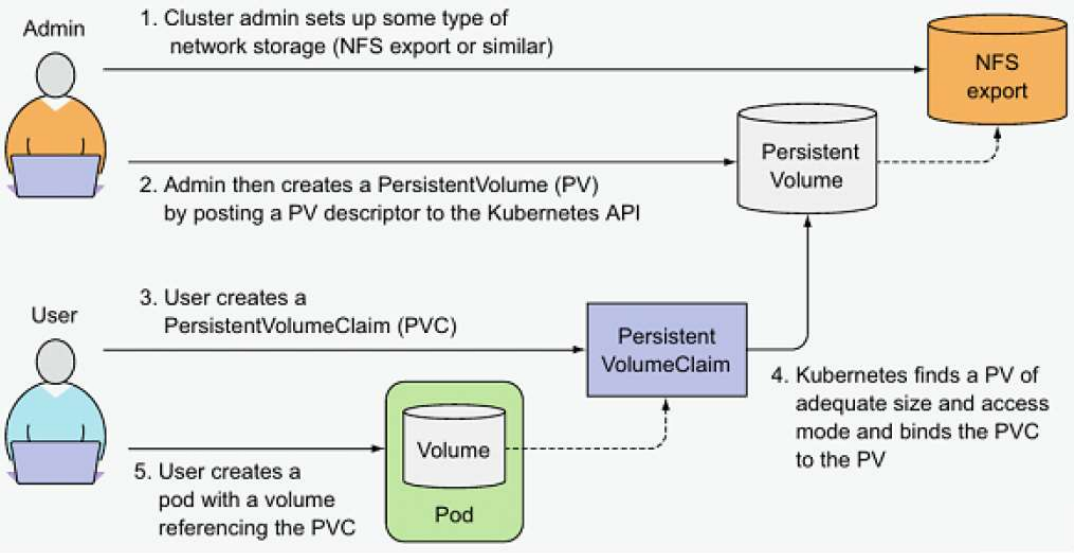
- 물리적인 디스크에 대한 정보와 논리적인 디스크에 대한 사용을 분리



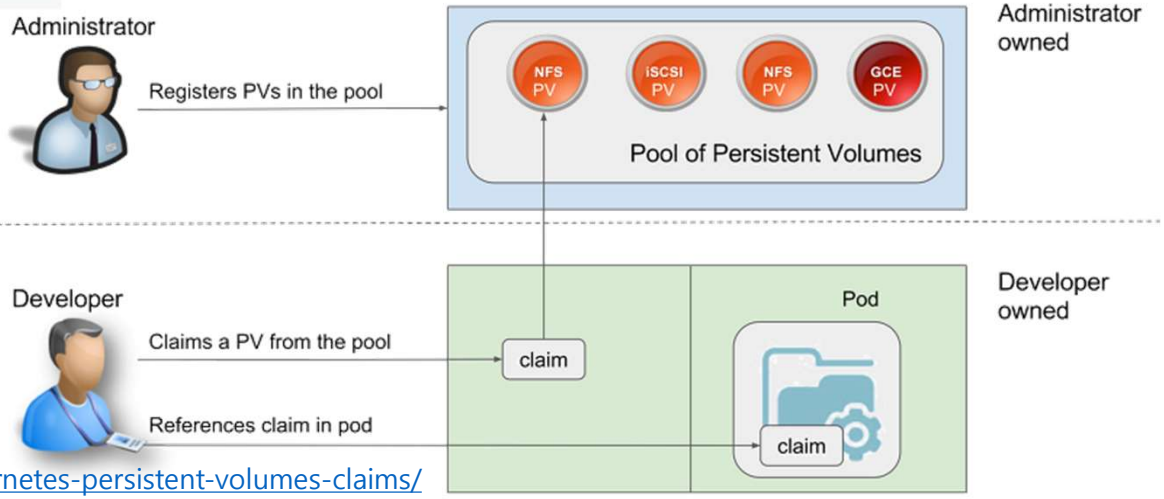
▲ 그림 6.8 GCE 퍼시스턴트 디스크를 직접 사용하는 경우와 PVC와 PV를 사용하는 경우

※ 참고 : https://thenewstack.io/strategies-running-stateful-applications-kubernetes-persistent-volumes-claims/

PersistentVolume(PV) & PersistentVolumeClaim(PVC)



▲ 그림 6.6 클러스터 관리자가 퍼시스턴트볼륨을 프로비저닝하면 파드는 퍼시스턴트볼륨클레임을 통해 이를 사용한다.



※ 참고 : https://thenewstack.io/strategies-running-stateful-applications-kubernetes-persistent-volumes-claims/

PersistentVolume

Reclaim Policy (반환 정책)

구분	설명
Retain	수동 반환
Delete	삭제
Recycle	Deprecated

storageClassName

특정 클래스의 PV는
해당 클래스를 요청하는 PVC에만 바인딩될 수 있다.

storageClassName이 없는 PV에는 클래스가 없으며
특정 클래스를 요청하지 않는 PVC에만 바인딩할 수 있다.

accessModes

구분		설명
ReadWriteOnce	RWO	하나의 노드에서 볼륨을 읽기-쓰기
ReadOnlyMany	ROX	여러 노드에서 볼륨을 읽기 전용
ReadWriteMany	RWX	여러 노드에서 볼륨을 읽기-쓰기

PersistentVolume.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume
  labels:
    release: stable
    env: staging

spec:
  storageClassName: manual
  persistentVolumeReclaimPolicy: Retain

  capacity:
    storage: 1Gi

  accessModes:
    - ReadWriteOnce

  hostPath:
    path: "/tmp/data"
    type: DirectoryOrCreate
```

```
❯ kubectl apply -f PersistentVolume.yaml
persistentvolume/pv-volume created

❯ kubectl get persistentvolume -o wide
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE   VOLUMEMODE
pv-volume   1Gi        RWO            Retain           Available           manual                  42s   Filesystem
```

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/persistent-volumes

PersistentVolumeClaim

- selector

  . **matchLabels** : 볼륨에 이 값의 레이블이 있어야함

  . **matchExpressions** : 키, 값의 목록, 그리고 키와 값에 관련된 연산자를 지정하여 만든 요구 사항 목록

  . matchLabels 및 matchExpressions의 모든 요구 사항이 AND 조건이다. 일치하려면 모두 충족해야 한다

PersistentVolumeClaim.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-claim

spec:
  storageClassName: manual

  accessModes:
    - ReadWriteOnce

  resources:
    requests:
      storage: 500Mi

  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: env, operator: In, values: [staging]}
```

```
❯ kubectl apply -f PersistentVolumeClaim.yaml
persistentvolumeclaim/pv-claim created

❯ kubectl get persistentvolumeclaims -o wide
NAME       STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE   VOLUMEMODE
pv-claim   Bound    pv-volume   1Gi        RWO            manual         73s   Filesystem

❯ kubectl get persistentvolume -o wide
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE   VOLUMEMODE
pv-volume   1Gi        RWO            Retain           Bound    default/pv-claim   manual                  10m   Filesystem
```

추가 확인 필요

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/persistent-volumes

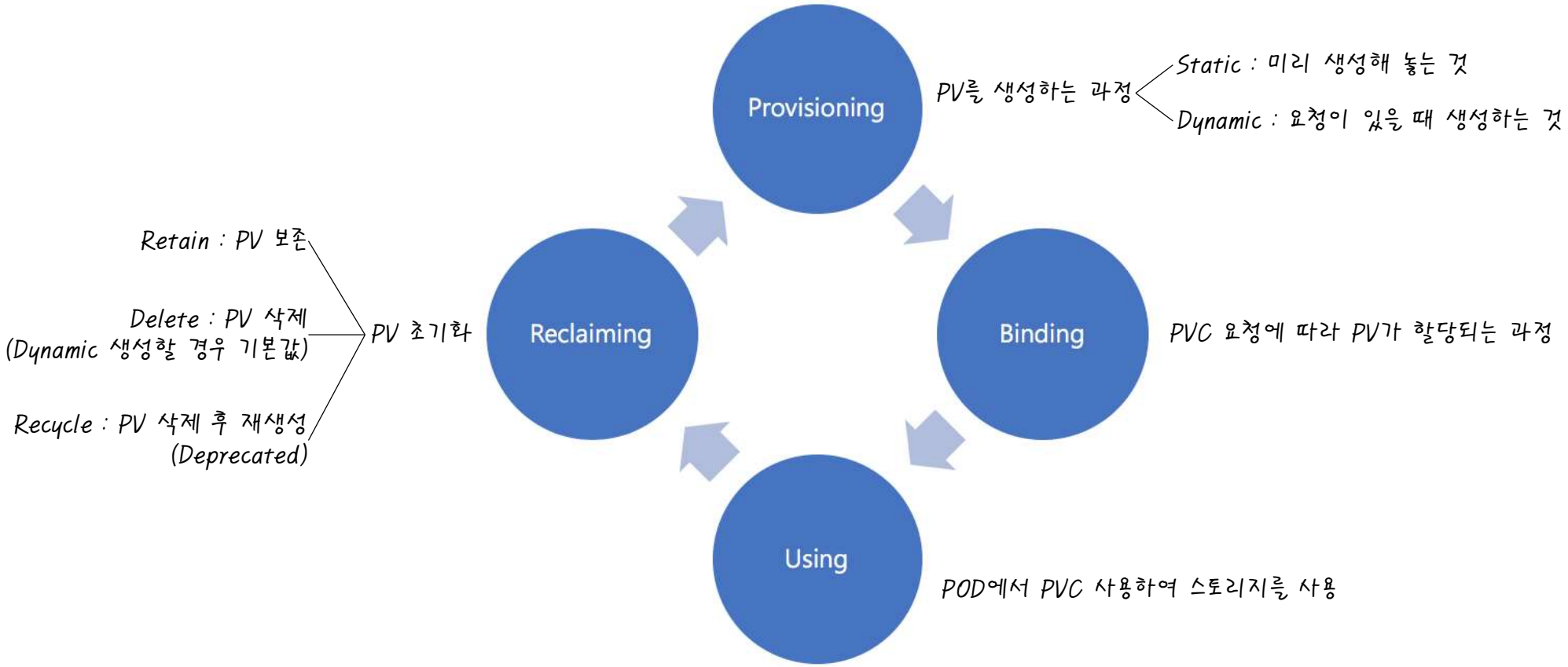
Claim

claim.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: claim

spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu

  template:
    metadata:
      labels:
        app: ubuntu

    spec:
      containers:
      - image: ubuntu:20.04
        name: worker
        command: ["/bin/sleep", "3650d"]
        volumeMounts:
        - name: claim-demo
          mountPath: /data/claim

      volumes:
        - name: claim-demo
          persistentVolumeClaim:
            claimName: pv-claim
```

```
❯ kubectl apply -f claim.yaml
replicaset.apps/claim created

❯ kubectl exec -it claim-n45fw -- ls -al /data/claim
total 8
drwxr-xr-x 2 root root 4096 May  7 20:16 .
drwxr-xr-x 3 root root 4096 May  7 20:16 ..

❯ kubectl exec -it claim-n45fw -- touch /data/claim/wow

❯ kubectl delete pods claim-n45fw
pod "claim-n45fw" deleted

❯ kubectl exec -it claim-4vt8z -- ls -al /data/claim
total 8
drwxr-xr-x 2 root root 4096 May  7 20:20 .
drwxr-xr-x 3 root root 4096 May  7 20:20 ..
-rw-r--r-- 1 root root    0 May  7 20:20 wow
```

“PV & PVC” Life Cycle (생명주기)



※ 참고 : https://arisu1000.tistory.com/27849

dynamic provisioning – 1/5

- [Static] 스토리지 볼륨 생성 → PersistentVolume 생성 → PersistentVolumeClaim 생성 → Volume Mount
- [Dynamic] StorageClass 생성 → PersistentVolume 생성 → PersistentVolumeClaim 생성 → Volume Mount

StorageClass-local.yaml

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer
```

```
❯ kubectl apply -f StorageClass-local.yaml
storageclass.storage.k8s.io/local-storage created

❯ kubectl get storageclasses -o wide
NAME            PROVISIONER                    RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-storage   kubernetes.io/no-provisioner   Delete          WaitForFirstConsumer   false                  12s
```

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/dynamic-provisioning/

dynamic provisioning – 2/5

PersistentVolume-local.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv

spec:
  capacity:
    storage: 1Gi

  accessModes:
  - ReadWriteOnce

  persistentVolumeReclaimPolicy: Retain

  storageClassName: local-storage

  local:
    path: /data/volumes/pv1

  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - worker2
```

```
[ in worker2 ]


> sudo mkdir –p /data/volumes/pv1
```

```
> kubectl apply -f PersistentVolume-local.yaml
persistentvolume/local-pv created

> kubectl get storageclasses -o wide
NAME            PROVISIONER                    RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-storage   kubernetes.io/no-provisioner   Delete          WaitForFirstConsumer   false                  2m47s

> kubectl get persistentvolume -o wide
NAME       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS    REASON   AGE   VOLUMEMODE
local-pv   1Gi        RWO            Retain           Available           local-storage            18s   Filesystem
```

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/dynamic-provisioning/

dynamic provisioning – 3/5

PersistentVolumeClaim-storageclass.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-pvc

spec:
  storageClassName: local-storage

  resources:
    requests:
      storage: 100Mi

  accessModes:
    - ReadWriteOnce
```

```
❯ kubectl apply -f PersistentVolumeClaim-storageclass.yaml
persistentvolumeclaim/local-pvc created

❯ kubectl get storageclasses -o wide
NAME            PROVISIONER                    RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-storage   kubernetes.io/no-provisioner   Delete          WaitForFirstConsumer   false                  8m17s

❯ kubectl get persistentvolume -o wide
NAME       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS    REASON   AGE     VOLUMEMODE
local-pv   1Gi        RWO            Retain           Available           local-storage            5m46s   Filesystem

❯ kubectl get persistentvolumeclaims -o wide
NAME        STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS    AGE   VOLUMEMODE
local-pvc   Pending                                      local-storage   36s   Filesystem
```

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/dynamic-provisioning/

dynamic provisioning – 4/5

dynamic.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet

metadata:
  name: dynamic

spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu

  template:
    metadata:
      labels:
        app: ubuntu

    spec:
      containers:
      - image: ubuntu:20.04
        name: worker
        command: ["/bin/sleep", "3650d"]
        volumeMounts:
        - name: dynamic-demo
          mountPath: /data/dynamic

      volumes:
        - name: dynamic-demo
          persistentVolumeClaim:
            claimName: local-pvc
```

```
❯ kubectl apply -f dynamic.yaml
replicaset.apps/dynamic created

❯ kubectl get storageclasses -o wide
NAME            PROVISIONER                    RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
local-storage   kubernetes.io/no-provisioner   Delete          WaitForFirstConsumer   false                  14m

❯ kubectl get persistentvolume -o wide
NAME       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM               STORAGECLASS    REASON   AGE   VOLUMEMODE
local-pv   1Gi        RWO            Retain           Bound    default/local-pvc   local-storage            12m   Filesystem

❯ kubectl get persistentvolumeclaims -o wide
NAME        STATUS   VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS    AGE     VOLUMEMODE
local-pvc   Bound    local-pv   1Gi        RWO            local-storage   6m52s   Filesystem

❯ kubectl get pods -o wide
NAME            READY   STATUS    RESTARTS   AGE   IP              NODE      NOMINATED NODE   READINESS GATES
dynamic-qkdx7   1/1     Running   0          47s   10.233.103.42   worker2   <none>           <none>
```

※ 참고 : https://kubernetes.io/ko/docs/concepts/storage/dynamic-provisioning/

dynamic provisioning – 5/5

```
❯ kubectl exec -it dynamic-qkdx7 -- ls -al /data/dynamic
total 8
drwxr-xr-x 2 root root 4096 May  7 21:41 .
drwxr-xr-x 3 root root 4096 May  7 22:32 ..

❯ kubectl exec -it dynamic-qkdx7 -- touch /data/dynamic/wow

❯ kubectl exec -it dynamic-qkdx7 -- ls -al /data/dynamic
total 8
drwxr-xr-x 2 root root 4096 May  7 22:36 .
drwxr-xr-x 3 root root 4096 May  7 22:32 ..
-rw-r--r-- 1 root root    0 May  7 22:36 wow
```

```
[ in worker2 ]

❯ ls -al /data/volumes/pv1
total 8
drwxr-xr-x 2 root root 4096  5월  8 07:36 .
drwxr-xr-x 3 root root 4096  5월  8 06:41 ..
-rw-r--r-- 1 root root    0  5월  8 07:36 wow
```

https://kahoot.it/