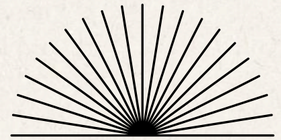
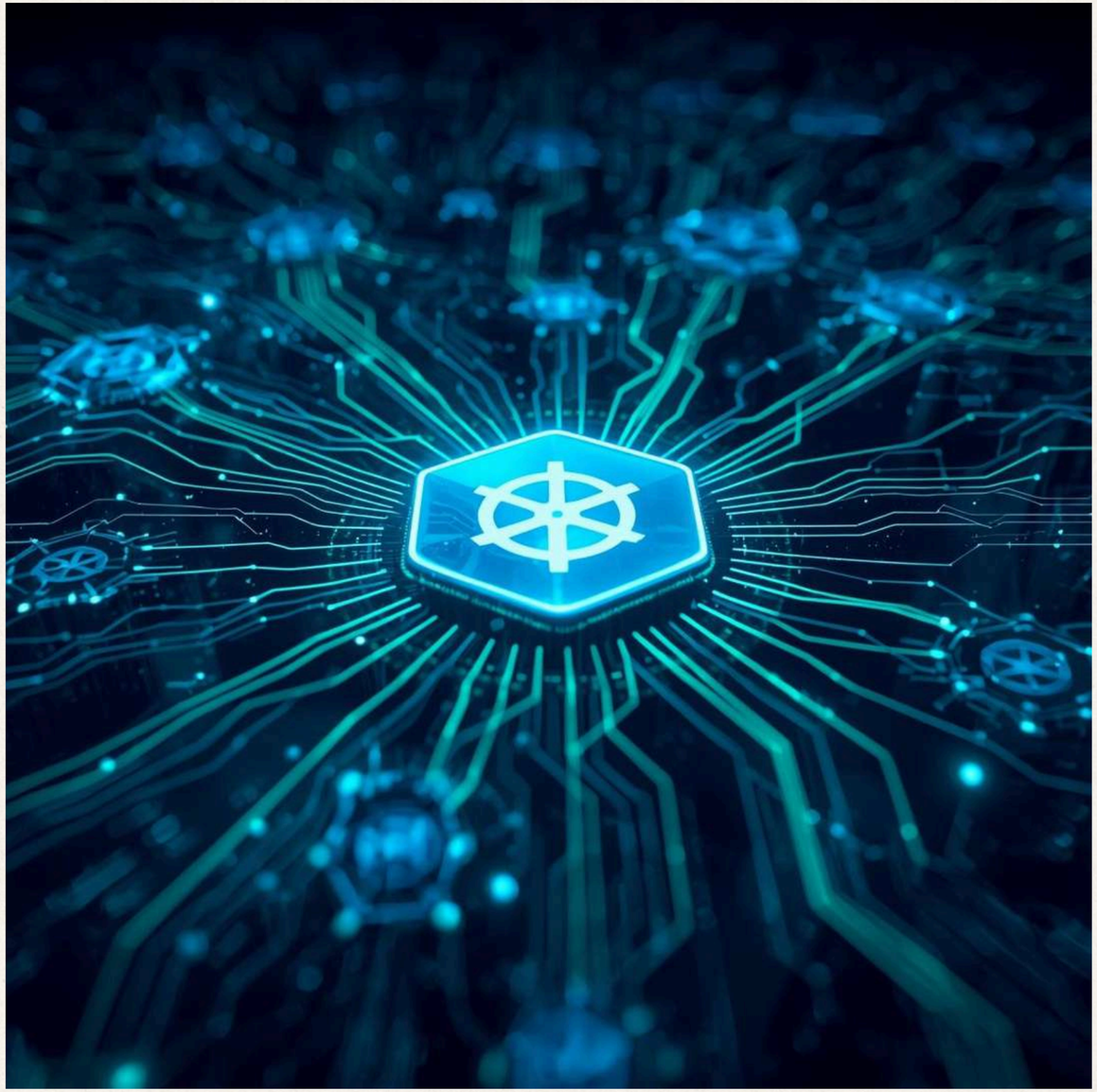
쿠버네티스 교과서

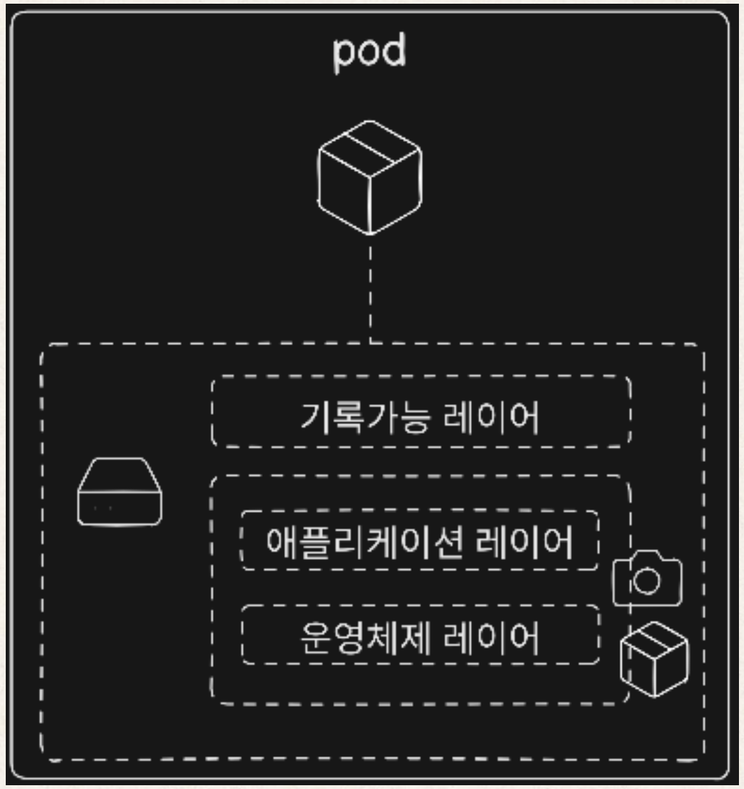
ch05





1 컨테이너 파일시스템
2 볼륨 마운트로 노드에 데이터저장
3 영구 볼륨과 영구볼륨클레임
4 스토리지 유형 & 동적 볼륨 프로비저닝
5 스토리지 선택시 고려할점
6 문제풀이

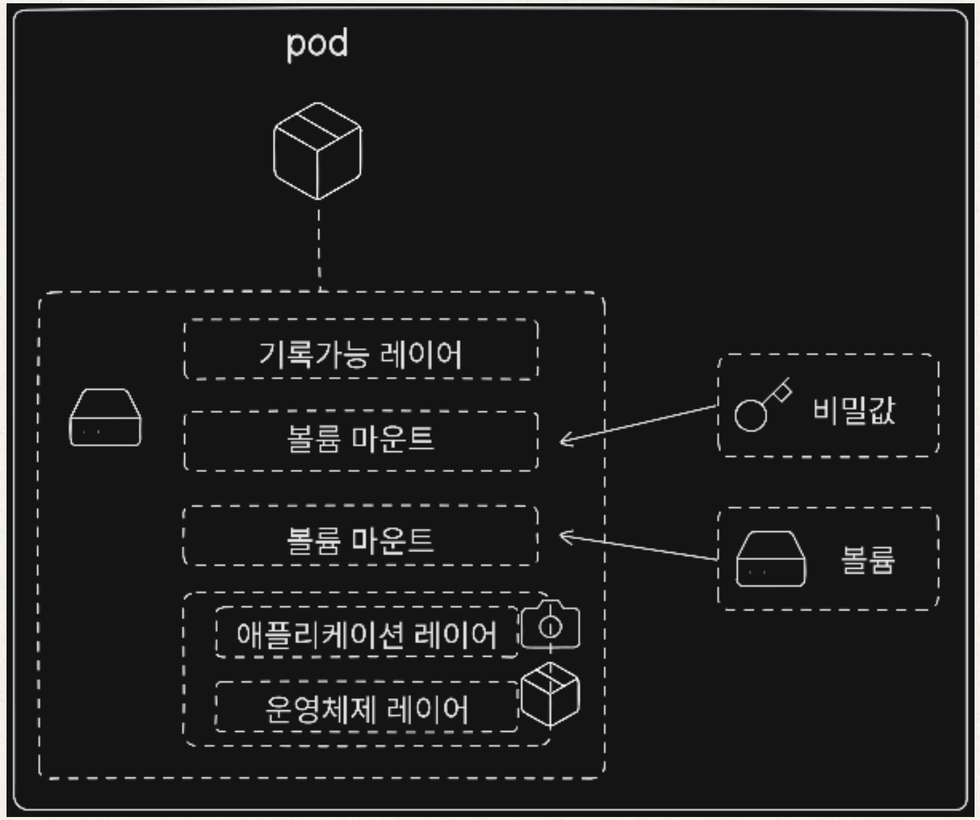
1. 컨테이너 파일시스템




```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f sleep/sleep.yaml
deployment.apps/sleep created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- sh -c 'echo ch05 > /file.
txt; ls /*.txt'
/file.txt
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pod -l app=sleep -o jsonpath='{.items[0].s
tatus.containerStatuses[0].containerID}'
containerd://7ff0fde118294d8fe16ba43c5ddc946d8c3c0210582ca082e3ab480af91c05c7ubuntu@local:~/files/
study/k8s/kiamol/ch05$ kubectl exec -it deploy/sleep -- killall5
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pod -l app=sleep -o jsonpath='{.items[0].s
tatus.containerStatuses[0].containerID}'
containerd://53941d26d287a90153d54fa915ce5a48fdf1ce33908b332c313bde3a2765b31cubuntu@local:~/files/
study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- ls /*.txt
ls: /*.txt: No such file or directory
command terminated with exit code 1
```

-컨테이너의 기록 가능 레이어에 기록한 데이터는 컨테이너 재시작시 사라진다.

1. 컨테이너 파일시스템




```yaml
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: data
          mountPath: /data #생성한 볼륨을 마운트할 컨테이너 경로
  volumes:
    - name: data
      emptyDir: {} #볼륨을 공디렉터리로 생성
```

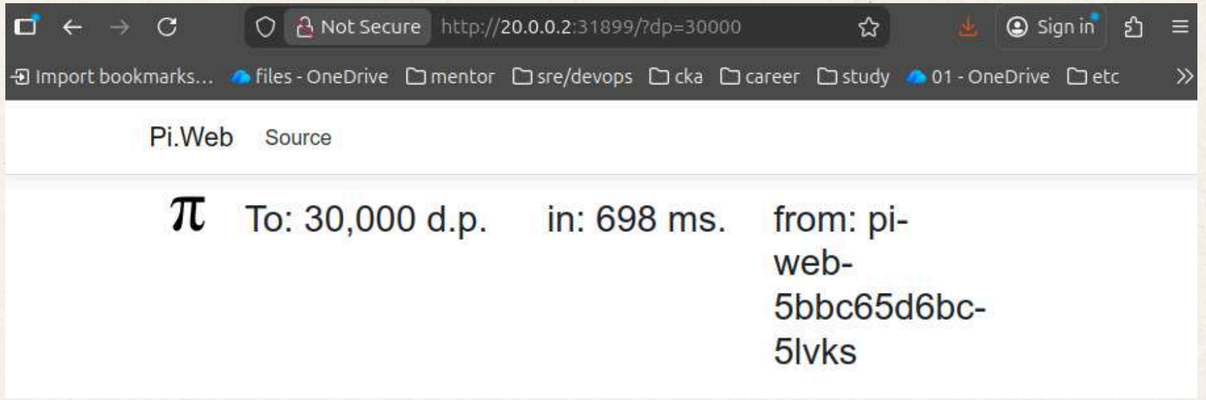
-공디렉터리는 컨테이너의 디렉터리에 마운트된다

공디렉터리
생성 시점: Pod 생성
삭제 시점: Pod 삭제
수명: Pod 생명주기와 동일

예) 로컬캐시

1. 컨테이너 파일시스템


```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f sleep/sleep-with-emptyDir.yaml
deployment.apps/sleep created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- ls /data
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- sh -c 'echo ch05 > /data/file.txt; ls /data'
file.txt
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pod -l app=sleep -o jsonpath='{.items[0].status.containerStatuses[0].containerID}'
containerd://aefa7e8d25779ea32813417c4cf23395ad43076ef05288ce94ebb0074c1fc131ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- killall5
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pod -l app=sleep -o jsonpath='{.items[0].status.containerStatuses[0].containerID}'
containerd://2ce31b1e032e9a14729b63bec03519282b9636e33b9279247738d1cbdd518723ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- cat /data/file.txt
ch05
ubuntu@local:~/files/study/k8s/kiamol/ch05$
```



-공디렉터리는 파드와 같은 생애 주기를 갖으므로, 볼륨에 저장된 데이터는 컨테이너가 재시작 되더라도 유지된다.

2.볼륨과 마운트로 노드에 데이터저장


```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f pi/v1/
configmap/pi-proxy-configmap created
service/pi-proxy created
deployment.apps/pi-proxy created
service/pi-web created
deployment.apps/pi-web created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl wait --for=condition=Ready pod -l app=pi-web
pod/pi-web-5bbc65d6bc-5lvks condition met
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get svc pi-proxy -o jsonpath='http://{.status.loadBalancer.ingress[0]
.*}:8080/?dp=30000'
http://:8080/?dp=30000ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/pi-proxy -- ls -l /data/nginx/cache
total 20
drwx------    5 nginx    nginx         4096 Jan 15 09:45 0
drwx------    3 nginx    nginx         4096 Jan 15 09:45 1
drwx------    4 nginx    nginx         4096 Jan 15 09:45 5
drwx------    3 nginx    nginx         4096 Jan 15 09:45 9
drwx------    3 nginx    nginx         4096 Jan 15 09:45 d
```

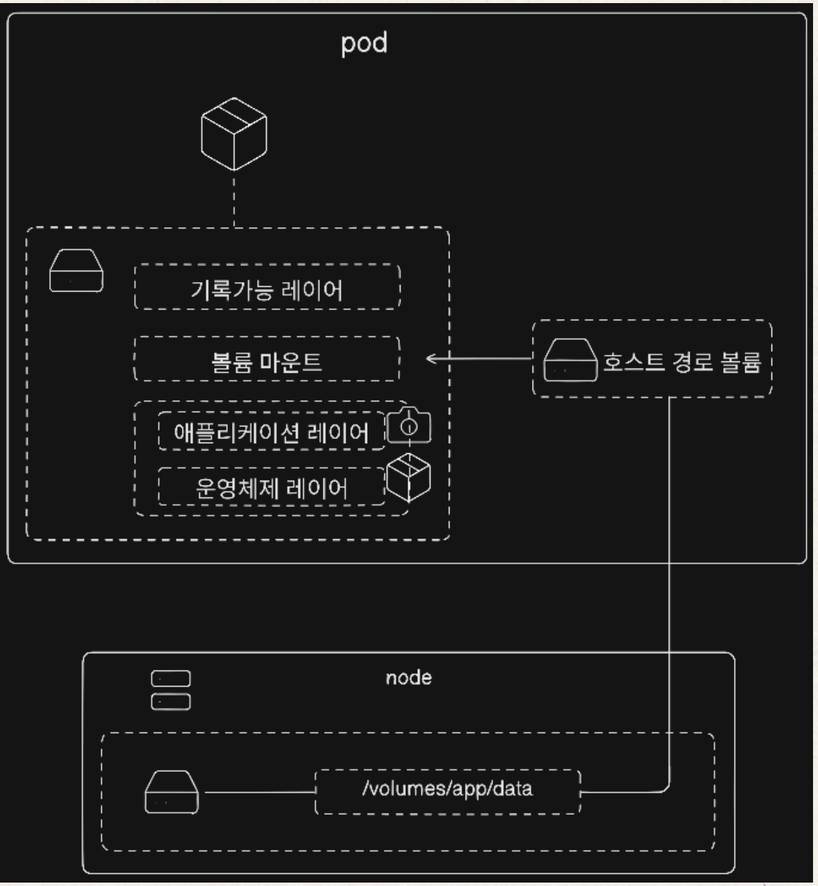
-pi-proxy파드로 요청을 보내면 공 디렉터리의 마운트된 디렉터리로
데이터를 캐시

```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl delete pod -l app=pi-proxy
pod "pi-proxy-7f7bd4586b-7rzfb" deleted from default namespace
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/pi-proxy -- ls -l /data/nginx/cache
total 0
```

-공디렉터리는 파드와 생명주기를 같이 하므로, 파드가 삭제되면
처음 상태의 빈 디렉터리가 됨

-공디렉터리 생성 위치
→ 파드가 생성된 노드의 디스크

2.볼륨과 마운트로 노드에 데이터저장




```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl delete pod -l app=pi-proxy
pod "pi-proxy-7f6c497486-4q9qm" deleted from default namespace
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/pi-proxy -- ls -l /data/nginx/cache
total 4
drwx------    3 nginx    nginx         4096 Jan 15 10:05 4
ubuntu@local:~/files/study/k8s/kiamol/ch05$
```

-파드를 삭제한 뒤에도 이전에 있었던 파일이 사라지지 않음

```
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: node-root
          mountPath: /node-root
  volumes:
    - name: node-root
      hostPath:
        path: /
        type: Directory
```

-호스트 경로 볼륨
→ 컨테이너 파일 시스템에 마운트하여 사용
→ 실제 데이터는 노드의 디스크에 기록

2.볼륨과 마운트로 노드에 데이터저장


```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f sleep/sleep-with-hostPath.yaml
deployment.apps/sleep created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- ls -l /var/log
total 0
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- ls -l /node-root/var/log |
head -n 3
total 230996
lrwxrwxrwx    1 root     root            39 Apr 23  2024 README -> ../../usr/share/doc/systemd/READM
E.logs
-rw-r--r--    1 root     root         36029 Jan  4 11:51 alternatives.log
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- whoami
root
```

-루트 디렉터리에 있는 다른 디렉터리에 접근 가능

-발생 가능한 문제점
→ 컨테이너가 노드의 루트 사용자에 의해서 실행되므로, 컨테이너는 노드의 모든 파일 시스템에 접근 가능

```
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: node-root
          mountPath: /pod-logs
          subPath: var/log/pods
        - name: node-root
          mountPath: /container-logs
          subPath: var/log/containers
  volumes:
    - name: node-root
      hostPath:
        path: /
        type: Directory
```

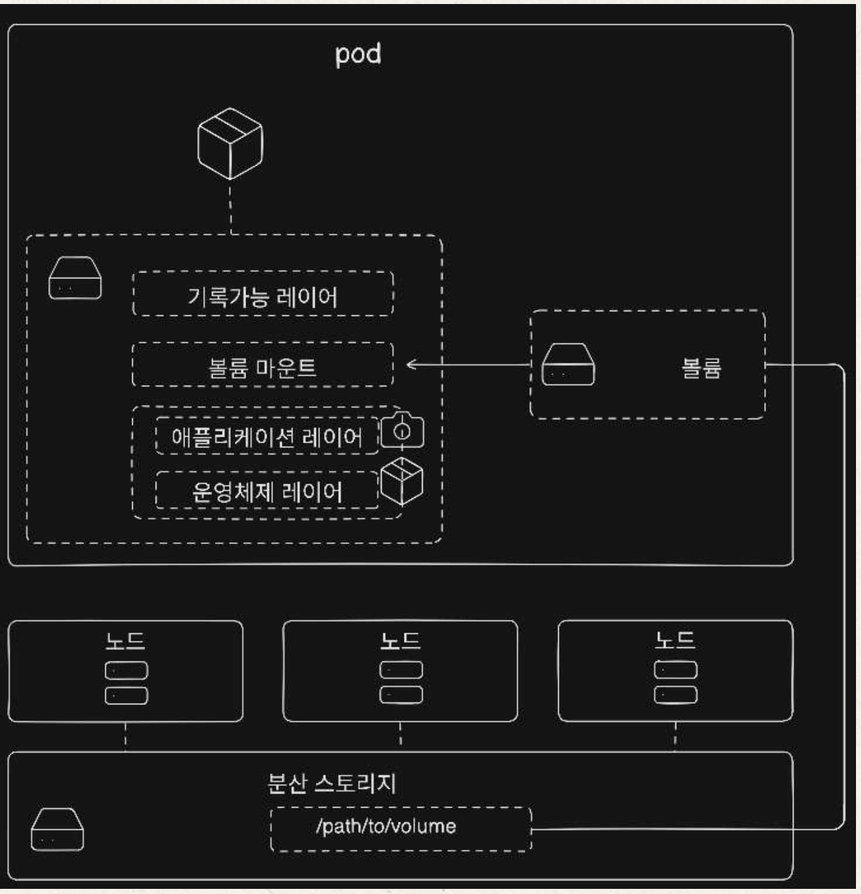
-컨테이너에 마운트시 호스트 디렉터리의 특정 경로를 컨테이너의 디렉터리에 마운트하여 다른 디렉터리에는 접근하지 못하도록 함

-해결방법
→ 볼륨마운트시 볼륨의 하위 디렉터리를 마운트
: 파일 시스템을 필요 이상으로 노출하지 않음

```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f sleep/sleep-with-hostPath-subPath.yaml
deployment.apps/sleep configured
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- sh -c 'ls /pod-logs | grep
_pi-'
default_pi-web-5bbc65d6bc-5lvks_ac9a9efd-bbaf-4711-abf5-35524fa2630d
```

-마운트된 호스트 디렉터리의 하위 디렉터리의 내용만 확인 가능

3.영구 볼륨과 영구볼륨클레임



-모든 노드가 같은 분산 스토리지에 연결도 가능

```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl label node k8s-worker-node-2
kiamol=ch05
node/k8s-worker-node-2 labeled
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get nodes -l kiamol=ch05
NAME                STATUS   ROLES    AGE    VERSION
k8s-worker-node-2   Ready    <none>   6d6h   v1.34.3
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f todo-list/persistent
Volume.yaml
persistentvolume/pv01 created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLAS
S   VOLUMEATTRIBUTESCLASS   REASON   AGE
pv01   50Mi       RWO            Retain           Available
    <unset>                          4s
```

-영구 볼륨을 사용하려면 영구볼륨 클레임이 필요
→ 쿠버네티스가 요청된 영구볼륨 클레임에 해당하는 영구볼륨을 찾아서 연결시켜준다.

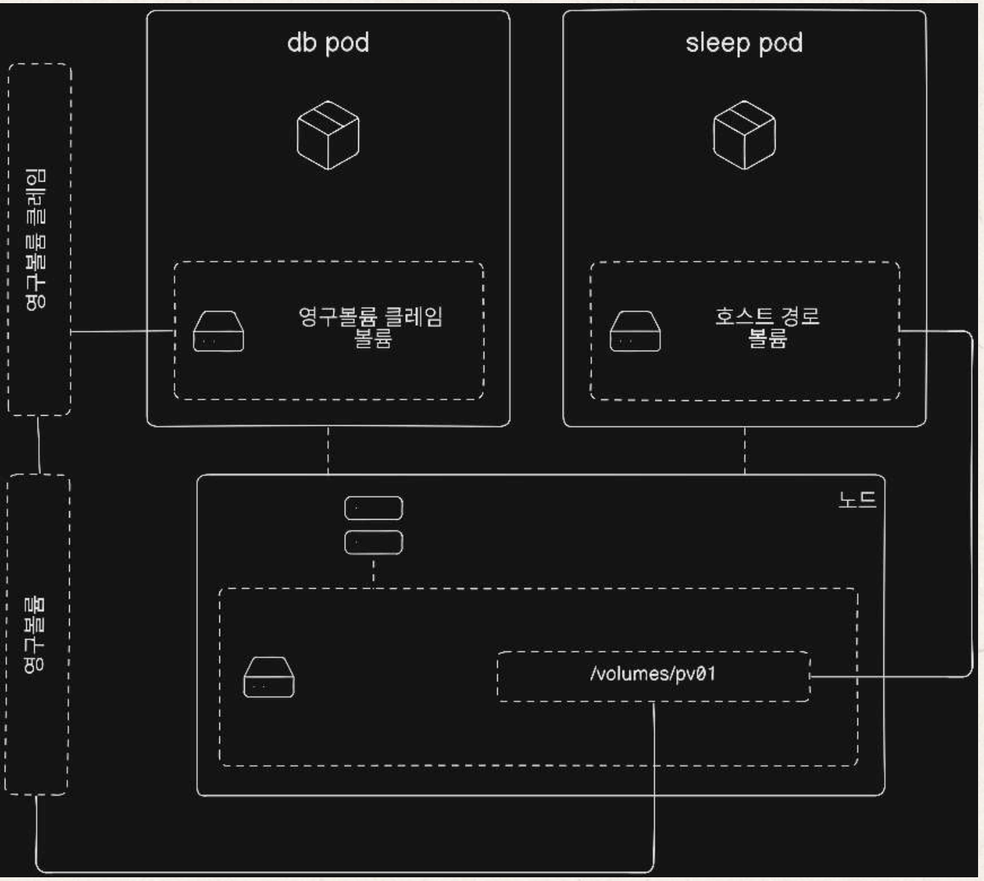
```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f todo-list/postgres-p
ersistentVolumeClaim.yaml
persistentvolumeclaim/postgres-pvc created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pvc
NAME           STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEA
TTRIBUTESCLASS   AGE
postgres-pvc   Bound    pv01     50Mi       RWO                           <unset>
                 3s
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
 STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pv01   50Mi       RWO            Retain           Bound    default/postgres-pvc
                <unset>                          2m9s
```

-영구볼륨만 있을때는 Available 상태

-영구볼륨클레임이 생성되면 Bound 상태로 변경된다.

-단점
→ 영구볼륨 클레임이 생성되었으나 요구사항과 일치하는 영구볼륨이 없을때 Pending 상태가 된다.

3.영구 볼륨과 영구볼륨클레임



-영구 볼륨과 영구볼륨 클레임의 관계도

-sleep pod 정의파일

```
spec:
  containers:
    - name: sleep
      image: kiamol/ch03-sleep
      volumeMounts:
        - name: node-root
          mountPath: /node-root
  volumes:
    - name: node-root
      hostPath:
        path: /
        type: Directory
```

-todo-db pod가 사용할 영구볼륨 정의

```
  name: pv01
spec:
  capacity:
    storage: 50Mi
  accessModes:
    - ReadWriteOnce
  local:
    path: /volumes/pv01
```

-db pod 정의파일

```
    volumeMounts:
      - name: secret
        mountPath: "/secrets"
      - name: data
        mountPath: /var/lib/postgresql/data

volumes:
  - name: secret
    secret:
      secretName: todo-db-secret
      defaultMode: 0400
      items:
        - key: POSTGRES_PASSWORD
          path: postgres_password
  - name: data
    persistentVolumeClaim:
      claimName: postgres-pvc
```

-영구볼륨 클레임을 리소스 정의 파일에서
볼륨으로 정의하여 컨테이너에 마운트

```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f sleep/sleep-with-hostPath.yaml
deployment.apps/sleep configured
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl wait --for=condition=Ready pod -l app=sleep
pod/sleep-68f5cd4db6-s6t56 condition met
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- mkdir -p /node-root/volum
es/pv01
ubuntu@local:~/files/study/k8s/kiamol/ch05$
```

-컨테이너에 마운트된 영구볼륨(호스트 경로)에
디렉터리 생성

3.영구 볼륨과 영구볼륨클레임


```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ k apply -f todo-list/postgres
/
secret/todo-db-secret configured
service/todo-db unchanged
deployment.apps/todo-db unchanged
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl logs -l app=todo-db -
-tail 1
2026-01-15 18:08:29.550 UTC [1] LOG:  database system is ready to accept
connections
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep --
sh -c 'ls -l /node-root/volumes/pv01 | grep wal'
drwx------    3 70       70             4096 Jan 15 18:08 pg_wal
```

-db 애플리케이션이 실행되고 볼륨에 데이터 파일을 생성

```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f todo-list/web
configmap/todo-web-config created
secret/todo-web-secret created
service/todo-web created
deployment.apps/todo-web created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl wait --for=condition=Ready pod -l app=todo-we
b
pod/todo-web-c8cfc568d-wzc64 condition met
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get svc todo-web -o jsonpath='http://{.status
.loadBalancer.ingress[0].*}:8081/new'
http://:8081/newubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl delete pod -l app=todo-web
error: name cannot be provided when a selector is specified
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl delete pod -l app=todo-db
pod "todo-db-7b8c68b76f-8kh9v" deleted from default namespace

ubuntu@local:~/files/study/k8s/kiamol/ch05$
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl exec deploy/sleep -- ls -l /node-root/volumes
/pv01/pg_wal
total 16388
-rw-------    1 70       70        16777216 Jan 15 18:15 000000010000000000000001
drwx------    2 70       70            4096 Jan 15 18:08 archive_status
ubuntu@local:~/files/study/k8s/kiamol/ch05$
```

-db 애플리테이션을 사용하는 todo 애플케이션 실행 및 테스트
→ db 파드를 삭제하여도 db 파드에서 사용하는 영구볼륨의
데이터는 삭제되지 않음

4.스토리지 유형 & 동적 볼륨 프로비저닝


```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc-dynamic
spec:
  #작성하지 않으면 기본값이 적용된다.
# storageClassName: ""
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

-영구볼륨클레임을 영구볼륨없이 배치했을때
일어나는 일은 클러스터에 따라 다르다
:StorageClass의 provisioner 방식에
따라 다를수 있음

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: example-storage-class
  annotations:
    storageclass.kubernetes.io/is-default-class: "false"
provisioner: rancher.io/local-path
reclaimPolicy: Retain
volumeBindingMode: WaitForFirstConsumer
```

-storage class 주요 필드
→ provisioner
: 영구볼륨을 만드는 주체
→ reclaimPolicy
: pvc가 삭제되었을때
pv처리여부 설정
→ volumeBindingMode
: 영구볼륨 클레임 생성후
영구볼륨의 생성 시점을 설정

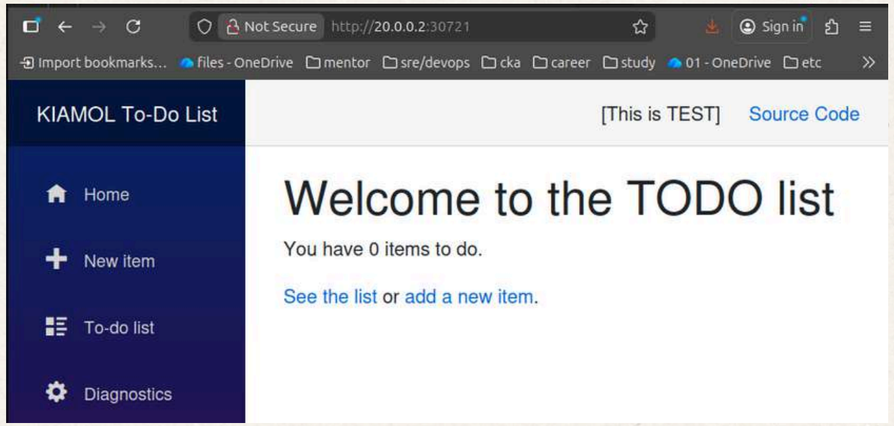
-동적 볼륨 프로비저닝
→ 간단한 방식
: 대부분의 k8s
플랫폼에서 사용

4.스토리지 유형 & 동적 볼륨 프로비저닝


```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ chmod +x cloneDefaultStorageClass.sh && ./cloneDefaultStorageClass.sh
configmap/clone-script created
pod/clone-sc created
pod/clone-sc condition met
storageclass.storage.k8s.io/kiamol created
configmap "clone-script" deleted from default namespace
pod "clone-sc" deleted from default namespace
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get sc
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
kiamol               rancher.io/local-path   Delete          WaitForFirstConsumer   false                  10s
local-path (default)   rancher.io/local-path   Delete          WaitForFirstConsumer   false                  8h
ubuntu@local:~/files/study/k8s/kiamol/ch05$
```

-kiamol이라는 storageclass를 새로 생성

```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f storageClass/postg
res-persistentVolumeClaim-storageClass.yaml
persistentvolumeclaim/postgres-pvc-kiamol created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl apply -f storageClass/todo-
db.yaml
deployment.apps/todo-db created
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pvc
NAME                  STATUS   VOLUME                                     CAPAC
ITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
postgres-pvc-kiamol   Bound    pvc-e557d0c7-d7e6-442e-98ed-cf5b273b2ed5   100Mi
      RWO            kiamol         <unset>                 11s
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM PO
LICY   STATUS   CLAIM                      STORAGECLASS   VOLUMEATTRIBUTESCL
ASS   REASON   AGE
pvc-e557d0c7-d7e6-442e-98ed-cf5b273b2ed5   100Mi      RWO            Delete
       Bound    default/postgres-pvc-kiamol   kiamol         <unset>
             7s
ubuntu@local:~/files/study/k8s/kiamol/ch05$ kubectl get pods -l app=todo-db
NAME                    READY   STATUS    RESTARTS   AGE
todo-db-79c6777-t6kst   1/1     Running   0          33s
```



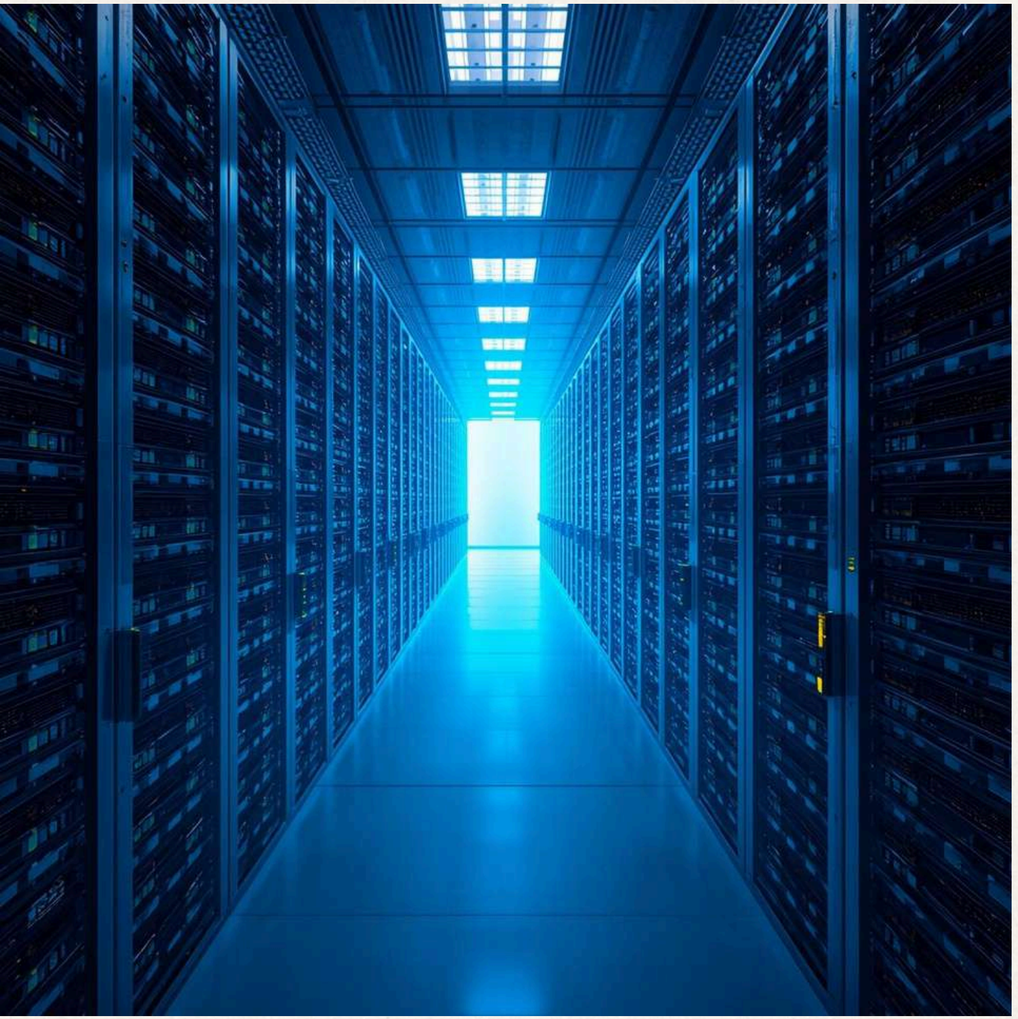
-마운트된 볼륨의 클레임이 변경되었으므로
이전 클레임의 볼륨에 있었던 데이터가
사라짐

-pod, pvc, pv 삭제 순서
: pod → pvc → pv 순서로
삭제

-동적 프로비전을 사용하는 storageclass를 사용하는 pvc 생성후 todo-db에 볼륨 마운트

-동적 프로비저닝이 작동하지 않을시
→ kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml

5.스토리지 선택시 고려할점

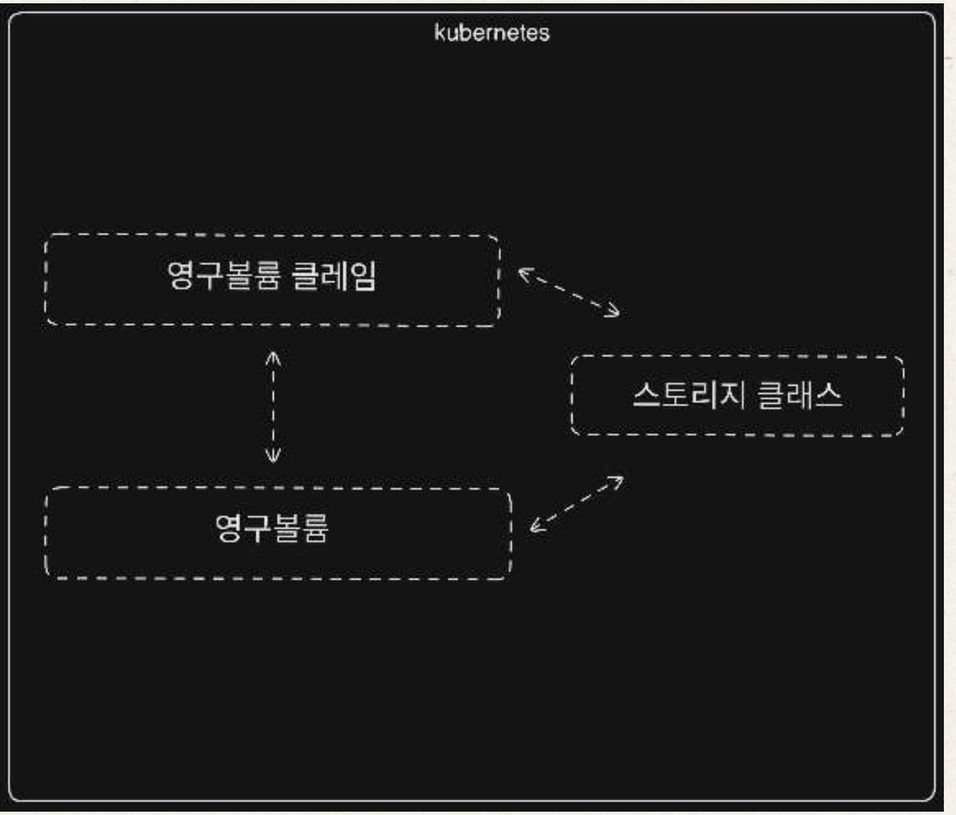


1.클라우드 환경 사용시 클러스터 전체에서 사용가능한
스토리지 솔루션에는 여러가지가 있다.

2.클라우드 환경의 매니지드 데이터베이스 서버는
데이터 백업, 스냅샷 등과 같은 기능을 제공한다.

3.데이터베이스 같은 유상태 애플리케이션은
반드시 쿠버네티스로 이전해야 하는것은 아니다.

결론



-pod의 볼륨 종류
→ 공디렉터리
→ hostPath
→ 영구볼륨 클레임 (pvc)

-pv와 pod volumes hostPath의 type 옵션
→ DirectoryOrCreate
: 디렉터리가 없으면 생성
→ Directory
: 경로에 디렉터리가 존재해야함

-StorageClass
→ pv
: 명시되어 있지 않아도
기본 StorageClass가 적용되지 않음
→ pvc
: 명시되어 있지 않으면 기본 StorageClass가
적용됨

-accessMode
→ ReadWriteOnce
: 하나의 노드에서
여러개의 pod이 접근 가능
→ ReadWriteMany
: 다수의 노드의 pod에서
접근가능

-영구볼륨 Reclaim Policy
→ Retain
: pvc 삭제후에 pv와 실제 디스크 유지
→ Delete
: pv와 실제 디스크 삭제

6.문제풀이


```
ubuntu@local:~/files/study/k8s/kiamol/ch05$ k logs todo-proxy-lab-7ccd8cbb9c-bh7qt
2026/01/16 07:32:03 [emerg] 1#1: mkdir() "/data/nginx/cache" failed (2: No such file or directory)
nginx: [emerg] mkdir() "/data/nginx/cache" failed (2: No such file or directory)
```

```
http {
  proxy_cache_path /data/nginx/cache

  server {
```

-nginx관련 디렉터리 문제인 것을 확인

```
volumes:
  - name: config
    configMap:
      name: todo-proxy-lab-configmap
  - name: config-directory
    persistentVolumeClaim:
        claimName: nginx-pv-claim
```

-pod정의 파일에 볼륨으로 pvc를 정의

```
volumeMounts:
  - name: config-directory
    mountPath: "/data/nginx/cache"
  - name: config
    mountPath: "/etc/nginx/"
    readOnly: true
```

-컨테이너에 첫번째로 volume 마운트

-진행기록

→ proxy deployments정의 파일의 문제인 것으로 생각

: deployment 파일문제 (공백등)x

->logs로 디렉터리에서 발생한 문제인 것을 확인

→ pod정의 파일에서 mkdir을 사용하여 컨테이너 디렉터리를 사용하려 하였으나 실패

→ proxy deployment에 volume 마운트를 하여 정상 실행이 되도록 함

6.문제풀이


```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: db-volume
  labels:
    type: local
spec:
  storageClassName: ""
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/opt/data2"
```

-db 영구볼륨 생성

```yaml
spec:
  containers:
    - name: web
      image: kiamol/ch04-todo-list
      env:
      - name: Database__Provider
        value: Sqlite
      - name: ConnectionStrings__ToDoDb
        value: "Filename=/data/todo-list-lab.db"
      volumeMounts:
      - name: db-volume
        mountPath: /data
  volumes:
  - name: db-volume
    persistentVolumeClaim:
      claimName: db-pv-claim
```

-pod 정의 파일에 마운트

Thank you