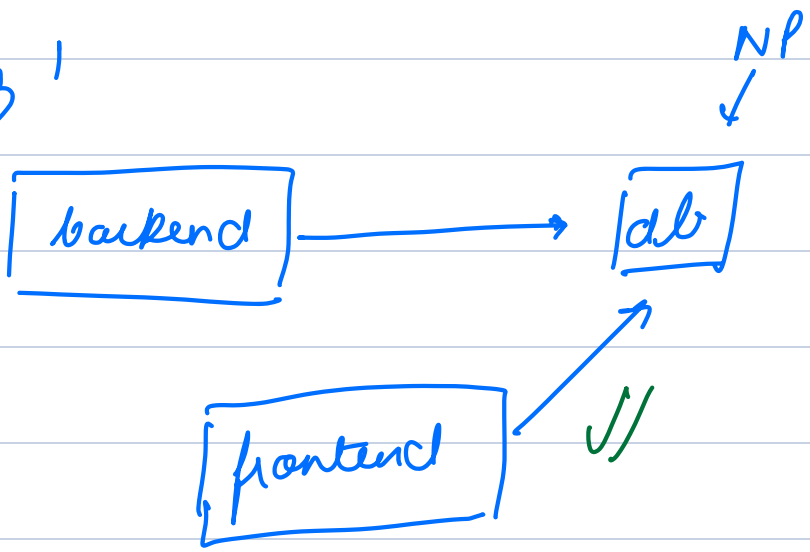
Kubernetes State Persistence

starts at 9:05pm

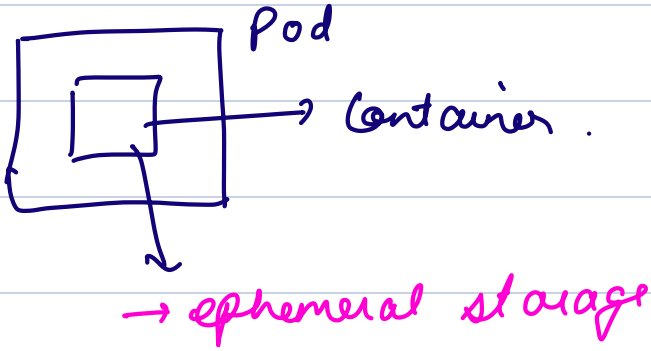
Agenda

1. Volumes in Kubernetes
2. Persistent Volumes and Persistent Volume Claims
3. Storage Classes
4. Stateful Sets
    1. Storage in Stateful sets

→ Quiz !



→ Volumes in K8s



container restarts ES → lost

pod rescheduled ES → lost

pod is deleted " "

to persist data → Volumes.

Types of Volumes

① EmptyDir { }

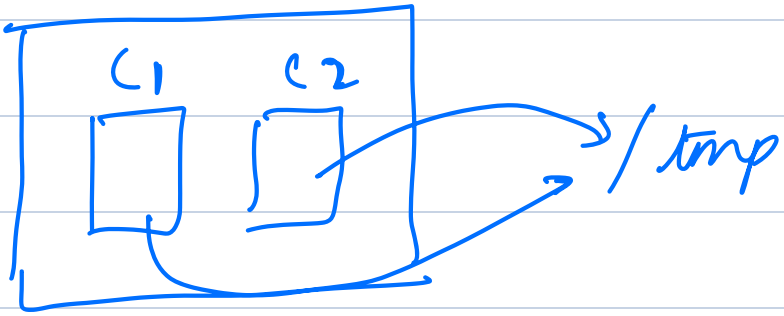
exists as long as the pod exists.

pod is deleted → data is lost.

Why emptyDir { }

① Persistence across container restarts.

② sharing data b/w containers.



③ Performance. → Fast storage.

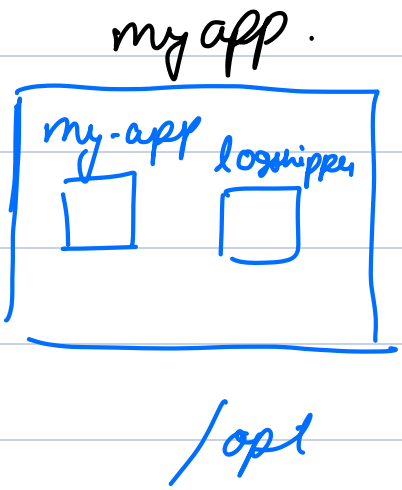
emptyDir: { medium: "Memory" }

emptyDir: {}

Feature	Writing to Container Filesystem	Using `emptyDir` Volume
Data Persistence	Lost when container restarts	Survives container restarts (but lost when pod is deleted)
Shared Storage	No, isolated per container	Yes, shared across containers in a pod
Performance	Uses container's disk storage	Can be **memory-backed** for high speed
Pod Deletion	Data is lost	Data is lost

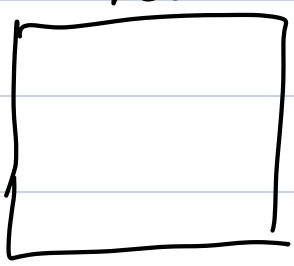
```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    app: myapp
spec:
  containers:
    - name: myapp
      image: alpine:latest
      command: ['sh', '-c', 'while true; do echo "logging $(date)" >> /opt/logs.txt; sleep 1; done']
```

```
    volumeMounts:
      - name: data
        mountPath: /opt
  - name: logshipper
    image: alpine:latest
    command: ['sh', '-c', 'tail -F /opt/logs.txt']
    volumeMounts:
      - name: data
        mountPath: /opt
volumes:
  - name: data
    emptyDir: {}
```



② Configmap and Secrets as Volumes.

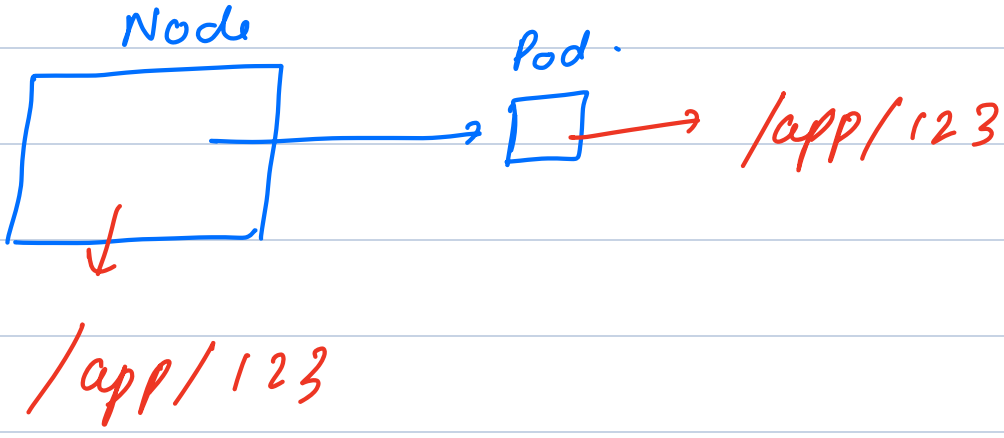
key 1 : value 1
key 2 : value 2.

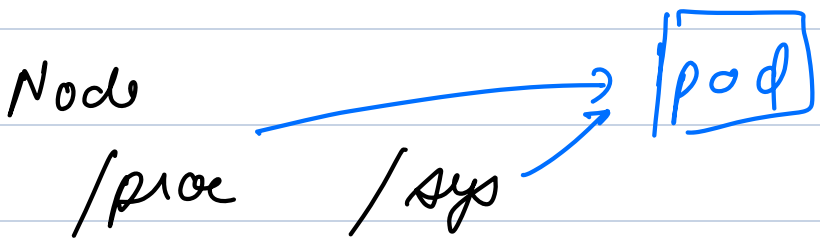


/tmp → mount path.
→ key 1 → file 1
→ key 2 → file 2

>

③ hostpath.





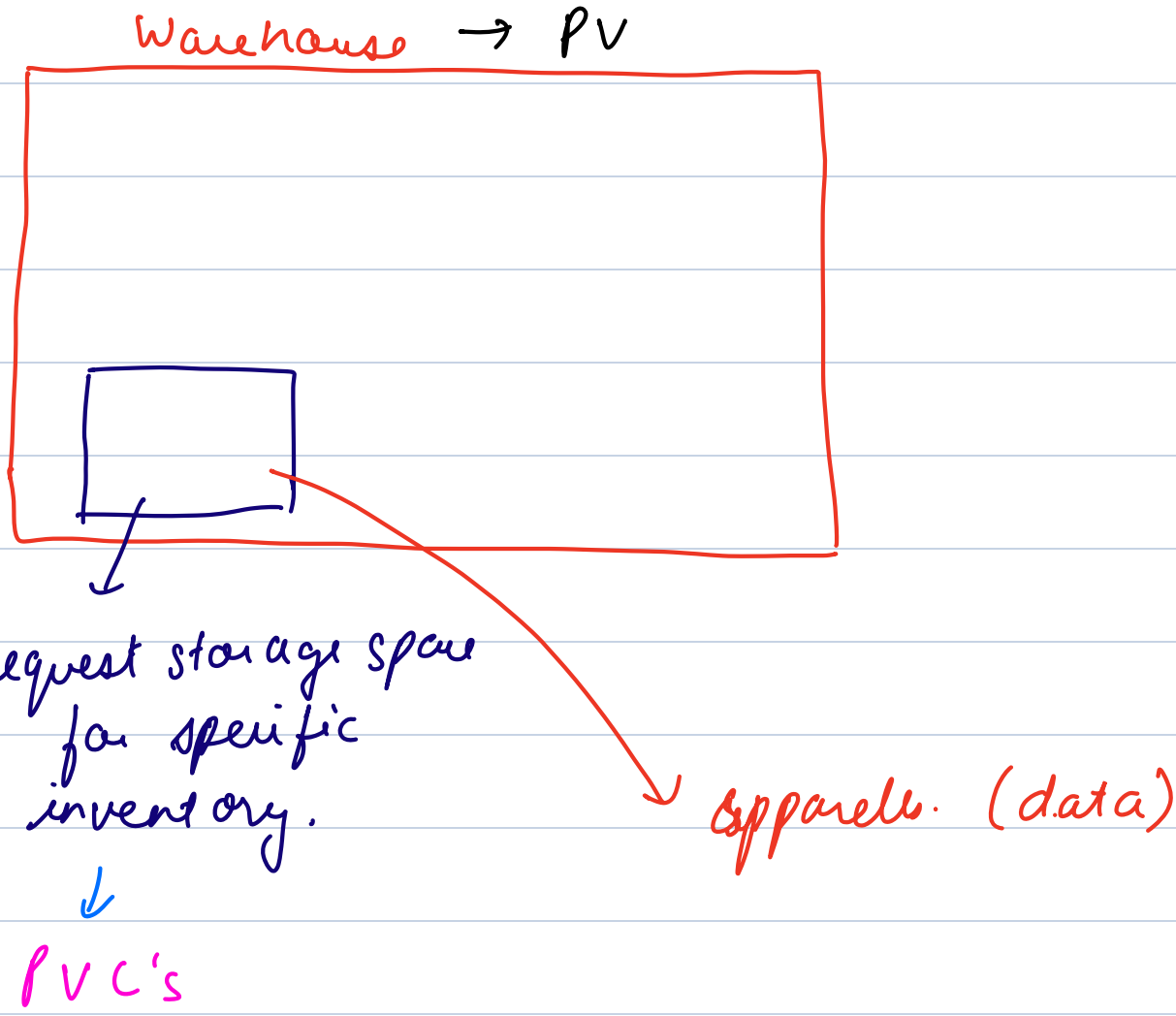

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-demo
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["/bin/sh", "-c"]
```

```
  args:
    - while true; do
        echo "$(date) - Log entry from pod" >> /var/log/app-logs/demo.log;
        sleep 5;
      done
  volumeMounts:
  - mountPath: /var/log/app-logs
    name: host-volume
volumes:
- name: host-volume
  hostPath:
    path: /var/log/app-logs   # Host directory to be mounted
    type: DirectoryOrCreate    # Creates directory if not exists
```

Limitations.

① Pods can modify the host files.
② Pods must be scheduled on the same node.
③ Breaks Portability.

Persistent volumes.



→ Pods can be rescheduled on other nodes.

A **Persistent Volume (PV)** is a **pre-provisioned storage resource** in the cluster, managed by an administrator.

→ storage capacity → 50Gi
→ access modes
→ reclaim policy.

→ Access Modes.

① Read Write Once (RWO)

- The volume can be mounted as read-write by a single node.

- **Only one pod** can use it at a time

② ReadOnly Many. (ROX)

- The volume can be mounted as read-only by many nodes.

- Multiple pods can **read** the volume at the same time

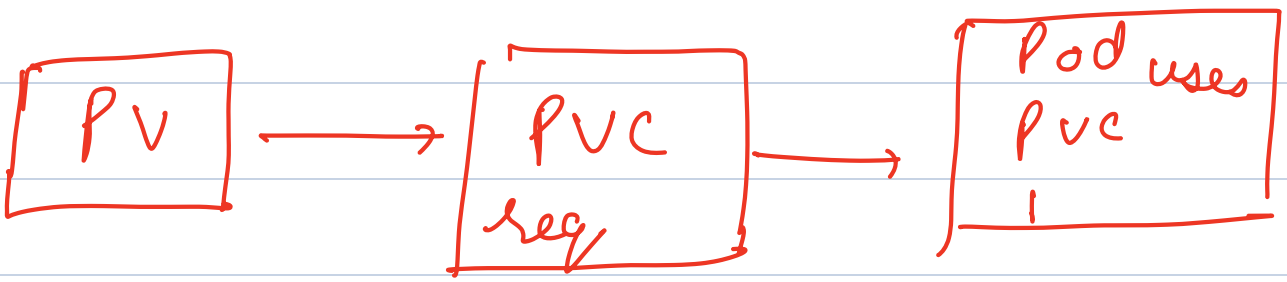
- No write access

③ Read Write Many (RWX)

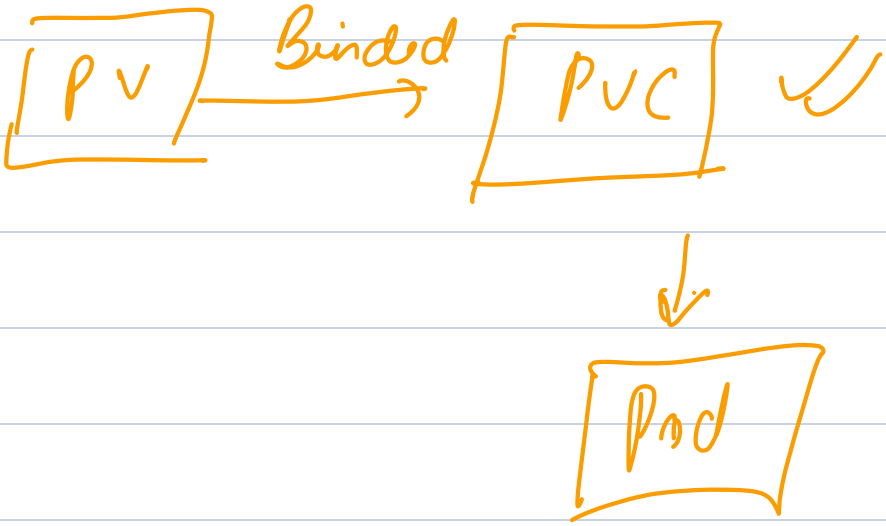
- Multiple pods can read and write simultaneously

- Works **across multiple nodes**

Access Mode	Symbol	Read/Write	Multiple Pods?	Multiple Nodes?
ReadWriteOnce	`RWO`	Read/Write	❌ No	❌ No
ReadOnlyMany	`ROX`	Read-Only	✅ Yes	✅ Yes
ReadWriteMany	`RWX`	Read/Write	✅ Yes	✅ Yes



Persistent Volume Claim.



Binding Process:

- When a PVC is created, Kubernetes looks for a PV that matches the PVC's request based on the storage class, size, and access modes.

- Once a suitable PV is found, it is bound to the PVC.

- This binding process makes the storage available to the Pod that requested it.

Reclaim Policies. →

① Retain

PVC → deleted

PV → retained

Released ?

② Delete.

PVC → deleted

PV → deleted.

EBS

Azure files

deleted.

Reclaim Policy	What It Does?	Common Use Cases
Retain (default)	Keeps the PV and its data, even if the PVC is deleted	Critical data (e.g., databases, logs)
Delete	Deletes the PV along with its storage (cloud disk, EBS, GCE)	Cloud storage (Amazon EBS, GCE Persistent Disks)

→ Demo (Read write Once)

Feature	ReadWriteOnce (RWO)	hostPath
Data Storage Location	Stored in a **Persistent Volume (PV)** (e.g., EBS, Azure Disk, etc.)	Stored directly on the **host node's filesystem**
Pod Scheduling	Kubernetes ensures the pod runs on the node where the PV is attached	Pod must be manually scheduled on the same node
Security & Isolation	More secure, managed storage	Less secure, pod can access critical host files
Persistence	Data persists even if the node fails (depending on storage backend)	Data is lost if the node is replaced

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-cloud
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  awsElasticBlockStore:
    volumeID: vol-1234567890abcdef
    fsType: ext4
```

Break → 10:40 pm.

→ Demo

Create a Persistent Volume with Retain Policy (`RWO`)

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-retain
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: "/mnt/data-retain" # This path is on your node
```

If this PV is used by a pod, the pod will see **the same files stored in `/mnt/data-retain` on the host**.

Create a Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim
```

```
metadata:
  name: pvc-retain
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Create a Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: pvc-test
spec:
  volumes:
    - name: storage
      persistentVolumeClaim:
        claimName: pvc-retain
  containers:
    - name: test-container
      image: busybox
```

```
command: ["sleep", "3600"]

volumeMounts:

 - mountPath: "/data"

   name: storage
```

→ Limitations of PV

1. Manual Provisioning.
2. Lack of Dynamic Allocation.
3. Waste of Resources.
4. Reclaim Policy limitation.

50 Gi → PV

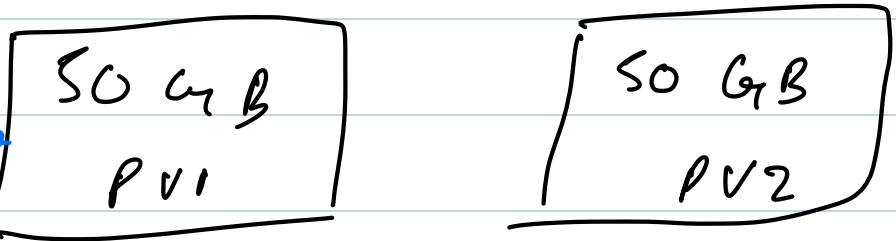
10 Gi → PVC

40 Gi → ?

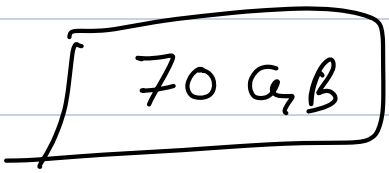
→ Storage Class

100 GB

PVC 40GB →



PVC



A **StorageClass** in Kubernetes provides a way to dynamically provision **Persistent Volumes (PVs)** on demand.

PV → ? No.

PVC → PV will be created

PVC 20GB ? → PV 20GB.

Key Components of storage Class.

1. **Provisioner** → Defines how Kubernetes interacts with the storage provider.

2. **Parameters** → Defines backend-specific options (like disk type, IOPS, etc.).

3. **Reclaim Policy** → What happens when a PVC is deleted? (`Retain`, `Delete`).

    - **`Retain`** → PV stays after PVC deletion. Manual cleanup required.

    - **`Delete`** → PV is deleted automatically when PVC is deleted.

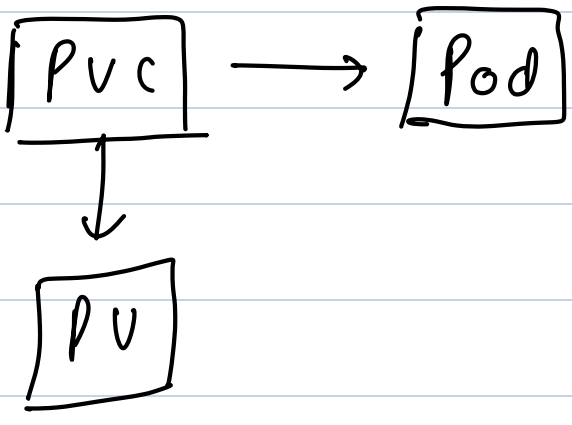
1. **Volume Binding Mode** → Controls when and how PVs are bound (`Immediate`, `WaitForFirstConsumer`).

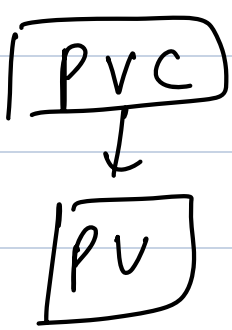
    1. When **`volumeBindingMode: Immediate`** is used, the **Persistent Volume (PV) is provisioned and bound to the PVC as soon as the PVC is created**, even if no pod is using it yet.

Provisioner	Storage Backend
`ebs.csi.aws.com`	AWS Elastic Block Store (EBS)
`pd.csi.storage.gke.io`	Google Persistent Disk
`disk.csi.azure.com`	Azure Disk
`nfs.csi.k8s.io`	NFS (Network File System)
`hostpath`	Host machine storage
rancher.io	Kind

→ immediate. (volume binding mode)

→ wait for first customer.





```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-storage
provisioner: ebs.csi.aws.com  # Provisioner for AWS EBS
parameters:
  type: gp3   # AWS EBS Volume Type
  iops: "3000"
  throughput: "125"
reclaimPolicy: Retain  # PVs remain even after PVC is deleted
volumeBindingMode: WaitForFirstConsumer  # PV is created only when a Pod uses the PVC
allowVolumeExpansion: true  # PVCs can request more storage later


apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: expandable-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
```

```
  requests:
    storage: 10Gi
  storageClassName: fast-storage
```

```
kubectl patch pvc expandable-pvc -p '{"spec":{"resources":{"requests":{"storage":"20Gi"}}}}'
```

- **File Storage (Default)**
  - `volumeMode: Filesystem` (default if not specified).
  - The PV is formatted with a filesystem (e.g., ext4, xfs).
  - Pods access it via a **file system interface**.
  - Examples: **NFS, EFS (AWS), CephFS, HostPath**.
- **Block Storage**
  - `volumeMode: Block`.
  - The PV is **raw block storage** without a filesystem.
  - Pods access it **as a raw device** (like `/dev/xvdb`).
  - Examples: **AWS EBS, GCE Persistent Disk, OpenStack Cinder**.

Reclaim Policy → Delete.
volumeBindingMode → WaitForFirstCustomer.

3 Gi

PV ? PVC ?

↓

Pod is deleted

↓ PV → deleted.

PV will remain

PVC → pending.

StorageClass Demo.

Storage Class

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: my-storage

provisioner: rancher.io/local-path # correct provisioner for Kind

reclaimPolicy: Retain

volumeBindingMode: WaitForFirstConsumer

PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
  storageClassName: my-storage
```

Pod will use the pvc which will create the PV

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  volumes:
    - name: storage-volume
      persistentVolumeClaim:
        claimName: my-pvc
```

```
containers:
  - name: app
    image: busybox
    command: [ "sleep", "3600" ]
    volumeMounts:
      - mountPath: /data
        name: storage-volume
```