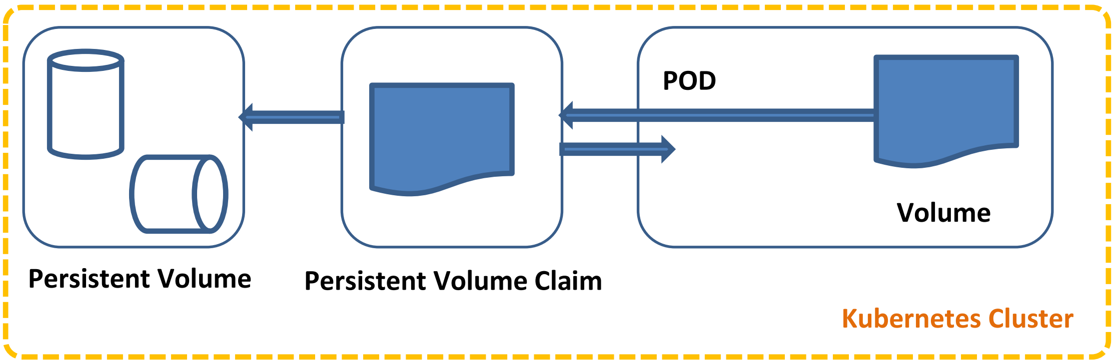
Configuring, Managing and Scheduling Kubernetes Storage

Persistent storage and containers:

The containers are ephemeral in nature. Container writable layer is destroyed when a container is deleted. When a POD is deleted, its container(s) is deleted from the node, so if there is any data written / stored inside the container in a POD, it will get lost with deleting the POD.

Storage API objects in Kubernetes

1) **Volume** : This is the actual storage, like AWS ealstic storage, or a folder. This is defined inside the POD spec.
2) **Persistent Volume (PV):** This is the actual storage available inside the cluster for the use inside a POD.
3) **Persistent Volume Claim (PVC):** Persistent Volume claim is the actual request made by the user for using the PersistentVolume inside a POD. Without having a PVC, one can not use the PV inside a POD.
4) **Storage Class:** this can be seen as group of storage for different storages (PVs) available inside kubernetes cluster.



Volume:

a. Persistent storage deployed as part of the POD specification in the POD manifest along with technical specification of the storage, like if it is NFS, we will have to provide NFS server DNS name and path.
b. The volume declaration also have to be provided with access information for the storage. This can pose few challenges.
    a. The code (manifest) for the POD can not be made portable as the technical details for the volume have to be defined inside the POD spec, making it not easy for portability between diff environments.
    b. Volumes also have the same lifecycle like the POD. So if a POD is removed / deleted, then the volume also gets deleted.
    c. To over come these challenges, we use the PVC that is easily portable as well.

Persistent Volume:

Persistent volume is defined by the cluster administrators and is mapped to the actual storage, like NFS, Cloud storage, storage disk etc.. So it is responsibility of the administrator to create or delete the storage **API object.** In the Persistent Volume API object defnitions, we will have to provide technical specification of the storage, like NFS server DNS, Path etc.

As this is independent object, if the POD is deleted the PV still lives on, it is persistent in nature.

The actual storage to be mapped on the node is managed by the Kubelet service.

Check below link for more detailed information on PVs.

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

Types of Persistent volumes.

These can be broadly classified into, Networked storage, Block and Cloud storage. Some sample types of PVs are as listed below, but there are lot more to this list.

- **Networked: NFS, azurefile**
- **Block: Fibre Channel, iSCSI**
- **Cloud: awsElasticBlockStore, azuredisk, gcePersistentDisk**

Check below link for more details.

Persistent Volumes | Kubernetes

Persistent Volume Claims

The Persistent Volume Claim is an API object which is basically a request by a user for storage to be used inside a POD. The PVC is defined with some technical specification like, Size, Access Mode (readWriteOnce, readWriteMany, readOnlyMany), Storage Class.

Using PVC add portability of the application configuration inside a POD.

When we deploy the PVC object, the cluster will map the PVC to an existing PV in static provisioning of volume depending upon the size, access mode and class requested.

Access Modes in PV:

The access mode defines how a persistent volume will be accessed by multiple Nodes or a Node. Here we say node because a Persisnt Volume is mapped to a Node or Nodes with deployment of the the PV API object.

1) ReadWriteOnce (RWO)
2) ReadWriteMany (RWX)
3) ReadOnlyMany (ROX)
4) ReadWriteOncePod (RWOP). if we want only one POD in the cluster to allow accessing the PV.

In case if the actual storage has a different access characteristics like if the NFS has read only charactiristic then this setting will superseed the API object access mode setting.

Static and Dynamic Provisioning of Persistent Volumes:

Workflow:

Create a **PersistentVolume** → Create a **PersistentVolumeClaim** → define **Volume** in **POD Spec**.

Defining all the objects manually is what the static provisioning is. In case of Dynamic provisioning of PV, A PersistentVolume is not deployed manually but is created automatically at the time of mapping a PVC in a POD definition, if the PV is not already present that matches the requested specification of a PV then the dynamic provisoining is used.

Storage Lifecycle in Kubernetes cluster:

Binding →	**Using** →	**Reclaim**
In the Binding process once a PVC is Created, the control loop find a PV that Matches PVC to PV, depending on size, access mode and class requested. If a PV is not present that matches to the specification, then the request goes into pending in state and it binds to the PV whenever a PV is available.	Once the binding is done the PVC and PV mapped till PVC's lifetime. Using the PVC is done by using the PVC in a POD's specification and that stays till the lifetime of the POD.	As there is a one-to-one mapping of the PVC and PV, if the PVC gets deleted, the PV can be deleted (often incase of dynamic provisioning of PV) or retained. Once a PVC is deleted the PV can be reused / Reclaims as specified in the PV reclaim policy. **Reclaim policy** options can be **Delete** (Default), **Retain** or **Recycle.**

Defining a Persistent Volume:

nfs-pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-data
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
```

```
    path: “/export/volumes/pod”
    server: 172.17.0.2
```

Defining Persistent Volume Claim

In a PVC definition there multiple specifications under which a PV gets mapped. acessMode, resources, storageClassName. Selector.

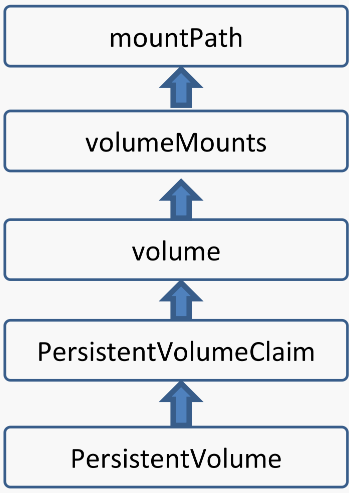
nfs-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nfs-data
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Using the Persistent Volume in PODs:

nfs-https-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  volumes:
    - name: website
      persistentVolumeClaim:
        claimName: pvc-nfs-data
  containers:
    - name: myfrontend
      image: httpd:latest
      volumeMounts:
      - name: website
        mountPath: "/var/www/html"
```



LAB: environment (Ubuntu 18.04, VirtualBox VMs, 2vCPUs, 2GB RAM, 50 GB. SWAP disabled)

Demo:

1) Storage Server overview
2) Static provisining Persistent Volume
3) Storage Lifecycle and reclaims Policy.

Setting up the NFS server:

```
$ sudo apt-get install nfs-kernel-server
```

```
$ sudo mkdir /export/volumes
$ sudo mkdir /exports/volumes/pod
```

Now to configure the NFS server, we have the config file at `/etc/exports.`

Let's run below command,

```
$ sudo bash -c 'echo "/export/volumes *(rw,no_root_squash,no_subtree_checck)" > /etc/exports'
$ cat /etc/exports
$ sudo systemctl restart nts-kernel-server.service
```

Now, In order the NFS export to be accessed on the cluster nodes we will also have to install the `nfs-common` (client package).

```
$ sudo apt-get install nfs-common
```

Now, check if we can mount the NFS server share on the node.

```
$ sudo mount -t nfs4 <nfs-sever-dns>:/export/volumes /mnt/
$ mount | grep nfs
$ sudo umount /mnt
```

Now let's start with deploying the YAML files starting with deploying the PV.

```
$ kubectl apply -f nfs-pv.yaml
$ kubectl get PersistentVolume pv-nfs-data
```

To get more details,

```
$ kubectl describe PersistentVolume pv-nfs-data
```

Now that we have the PV created, let's deploy the PVC,

```
$ kubectl apply -f nfs-pvc.yaml
$ kubectl get PersistentVolumeClaim pvc-nfs-data
```

Check the status of PV which show the PV is bound

To get more details.

```
$ kubectl describe PersistentVolumeClaim pvc-nfs-data
```