Kubernetes

PersistentVolume:

In production, multiple pods are created, and each pod may require storage to persist data. To provide storage for pods, PersistentVolumes (PVs) come in handy. A PersistentVolume is a piece of storage allocated in the cluster, which can be provisioned with specific storage capacities, such as 1Gi, 100Gi, etc., and made available to pods.
The location of this storage is in the worker node at a given path.

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: local-pv  #Name for pv
 labels:
   app: local # label don’t forget to mention
spec:
 capacity:
   storage: 1Gi #How much storage you want to create
 accessModes:
   - ReadWriteOnce   #How you want to access the storage
 persistentVolumeReclaimPolicy: Retain   #what you want after completion of task
 storageClassName: local-storage  #which kind of storage you want
hostPath:
   path: /mnt/data  #where you want storage location
```

AccessModes:

This shows how you want to access data. There are four types of access mode.

ReadWriteOnce - One pod and one Node access
ReadonlyMany - multiple node access for read only
ReadWriteMany - Multiple pods on different node
ReadWriteOncePod - For exclusive single pod access

ReclaimPolicy

Retain - The PV is not deleted when released. Instead, it is retained for manual intervention by administrator

Delete - The underlying storage resource is automatically deleted when the PV is released.

StorageClassName :

This is a term which tells the administrator where you want to save your data.

Local-storage - In local system
Cloud - Cloud storage like AWS, Azure or GCP
NFS - Network File system

PersistentVolumeClaim:

As we have created a PersistentVolume (PV), we need to claim it so that it can be used by a pod. This is done using a PersistentVolumeClaim (PVC), which allows users to request storage with specific characteristics, such as size, access modes, and storage classes. After the PVC is created, it binds to a compatible PV, and the storage becomes available for the pod to use.

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

How PV and PVC Bind Together

To successfully bind a **PersistentVolume (PV)** and **PersistentVolumeClaim (PVC)** in Kubernetes, the following three factors are taken into consideration:

1. Size

When binding a PV to a PVC, the Kubernetes cluster ensures that the volume provides the requested storage size, neither more nor less. For example, if both the PV and PVC specify a size of 1Gi, the binding will be successful. If the PVC requests more storage than the PV offers (or less), the binding will fail.

Example:

PV has capacity: 1Gi

PVC has resources.requests.storage: 1Gi
The PV and PVC will bind because their requested sizes match.

2. AccessModes

The **AccessModes** defined in both the PV and PVC must match for successful binding. Access modes determine how the volume can be accessed (e.g., ReadWriteOnce, ReadWriteMany, ReadOnlyMany).

If the **PVC** specifies an access mode that is incompatible with the available access modes of the **PV**, the binding will not happen.

Example:

PV has accessModes: [ReadWriteOnce]

PVC has accessModes: [ReadWriteOnce]
The PV and PVC will bind because they both allow read-write access from a single node.

3. StorageClassName

The **storageClassName** is a key factor for binding PV and PVC. A PVC can only bind to a PV that has the same storageClassName. This ensures that both the PV and PVC use the same type of storage (e.g., local-storage, standard, nfs, etc.).

If the storageClassName is not specified in the PVC, Kubernetes will attempt to bind it to any PV that has **no storageClassName** or the **default storageClassName**. If the PVC specifies a storageClassName, it must match a PV with the same storageClassName.

Example:

PV has storageClassName: local-storage

PVC has storageClassName: local-storage
The PV and PVC will bind because they have the same storageClassName.

Conclusion :

In today's article, we explored PersistentVolume (PV) and PersistentVolumeClaim (PVC). Additionally, we examined how PV and PVC are bound to each other.