**Persistent Storage- Openshift:**

**Persistent Volume (PV):**

* PVs are defined by a Persistent Volume API object, which represents a piece of existing networked storage in the cluster that has been provisioned by an administrator.
* It is a resource in the cluster just like a node is a cluster resource.
* PVs are volume plug-ins like Volumes, but have a lifecycle independent of any individual [pod](https://docs.openshift.com/container-platform/3.5/architecture/core_concepts/pods_and_services.html#pods) that uses the PV.
* PV objects capture the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.
* Each PV contains a spec and status, which is the specification and status of the volume.

Persistent Volume Object Definition

**apiVersion: v1**

**kind: PersistentVolume**

**metadata:**

**name: pv0003**

**spec:**

**capacity:**

**storage: 5Gi**

**accessModes:**

**- ReadWriteOnce**

**persistentVolumeReclaimPolicy: Recycle**

**nfs:**

**path: /tmp**

**server: 172.17.0.2**

**Persistent Volume Claims (PVC’s):**

* PVCs are defined by a Persistent Volume Claim API object, which represents a request for storage by a developer.
* It is similar to a pod in that pods consume node resources and PVCs consume PV resources.
* For example, pods can request specific levels of resources (e.g., CPU and memory), while PVCs can request specific [storage capacity](https://docs.openshift.com/container-platform/3.5/architecture/additional_concepts/storage.html#pv-capacity) and [access modes](https://docs.openshift.com/container-platform/3.5/architecture/additional_concepts/storage.html#pv-access-modes) (e.g, they can be mounted once read/write or many times read-only).
* Each PVC contains a **spec** and **status**, which is the specification and status of the claim.

*Persistent Volume Claim Object Definition*

**kind: PersistentVolumeClaim**

**apiVersion: v1**

**metadata:**

**name: myclaim**

**annotations:**

**volume.beta.kubernetes.io/storage-class: gold**

**spec:**

**accessModes:**

**- ReadWriteOnce**

**resources:**

**requests:**

**storage: 8Gi**

### Storage Class

* Claims can optionally request a specific StorageClass by specifying its name in the storageClassName attribute.
* Only PVs of the requested class, ones with the same storageClassName as the PVC, can be bound to the PVC.
* The cluster administrator can configure dynamic provisioners to service one or more storage classes. They create a PV on demand that matches the specifications in the PVC, if they are able.
* The cluster administrator can also set a default StorageClass for all PVCs. When a default storage class is configured, the PVC must explicitly ask for StorageClass or storageClassName annotations set to "" to get bound to a PV with a no storage class.

**How to Provision:**

* In response to requests from a developer defined in a PVC, a cluster administrator configures one or more dynamic provisioners that provision storage and a matching PV.
* Alternatively, a cluster administrator can create a number of PVs in advance, which carry the details of the real storage that is available for use by cluster users. PVs exist in the API and are available for consumption

### How to Bind:

* A user creates a PersistentVolumeClaim with a specific amount of storage requested and with certain access modes and optionally a StorageClass.
* A control loop in the master watches for new PVCs. It either finds a matching PV or waits for a provisioner for the StorageClass to create one, then binds them together.
* The user will always get at least what they asked for, but the volume might be in excess of what was requested. This is especially true with manually provisioned PVs. To minimize the excess, OpenShift Container Platform binds to the smallest PV that matches all other criteria.
* Claims remain unbound indefinitely if a matching volume does not exist or cannot be created with any available provisioner servicing a StorageClass.
* Claims are bound as matching volumes become available. For example, a cluster with many manually provisioned 50Gi volumes would not match a PVC requesting 100Gi. The PVC can be bound when a 100Gi PV is added to the cluster.

### How to Use:

* Pods use claims as volumes.
* The cluster inspects the claim to find the bound volume and mounts that volume for a pod. For those volumes that support multiple access modes, the user specifies which mode is desired when using their claim as a volume in a pod.

### How to Release

* When a user is done with a volume, they can delete the PVC object from the API which allows reclamation of the resource.
* The volume is considered "released" when the claim is deleted, but it is not yet available for another claim. The previous claimant’s data remains on the volume which must be handled according to policy.

**Note: Previously claimant’s data must be handled according to the policy else the volume is not in released state.**

### How to Reclaim

* The reclaim policy of a Persistent Volume tells the cluster what to do with the volume after it is released. Currently, volumes can either be ***retained*** or ***recycled***.
* Retention allows for manual reclamation of the resource. For those volume plug-ins that support it, recycling performs a basic scrub on the volume (e.g., rm -rf /<volume>/\*) and makes it available again for a new claims

### Capacity

* Generally, a PV will have a specific storage capacity. This is set using the PV’s **capacity** attribute.
* Currently, storage capacity is the only resource that can be set or requested. Future attributes may include IOPS, throughput, etc.

**Different access modes:**

* A PersistentVolume can be mounted on a host in any way supported by the resource provider. Providers will have different capabilities and each PV’s access modes are set to the specific modes supported by that particular volume.
* For example, NFS can support multiple read/write clients, but a specific NFS PV might be exported on the server as read-only. Each PV gets its own set of access modes describing that specific PV’s capabilities.
* Claims are matched to volumes with similar access modes. **The only two matching criteria are access modes and size**. A claim’s access modes represent a request. Therefore, the user may be granted more, but never less. For example, if a claim requests RWO, but the only volume available was an NFS PV (RWO+ROX+RWX), the claim would match NFS because it supports RWO.

**Rules for matching:**

* Direct matches are always attempted first.
* The volume’s modes must match or contain more modes than you requested. The size must be greater than or equal to what is expected.
* If two types of volumes (NFS and iSCSI, for example) both have the same set of access modes, then either of them will match a claim with those modes.
* There is no ordering between types of volumes and no way to choose one type over another.
* All volumes with the same modes are grouped, then sorted by size (smallest to largest). T
* he binder gets the group with matching modes and iterates over each (in size order) until one size matches.

**Types of access modes:**

| **Access Mode** | **CLI Abbreviation** | **Description** |
| --- | --- | --- |
| ReadWriteOnce | RWO | The volume can be mounted as read-write by a single node. |
| ReadOnlyMany | ROX | The volume can be mounted read-only by many nodes. |
| ReadWriteMany | RWX | The volume can be mounted as read-write by many nodes |

**Errors that may occur:**

* A volume’s **AccessModes** are descriptors of the volume’s capabilities. They are not enforced constraints. The storage provider is responsible for runtime errors resulting from invalid use of the resource.
* For example, a GCE Persistent Disk has **AccessModes** **ReadWriteOnce** and **ReadOnlyMany**. The user must mark their claims as **read-only** if they want to take advantage of the volume’s ability for ROX. Errors in the provider show up at runtime as mount errors.
* Error🡪iSCSI and Fibre Channel volumes do not have any fencing mechanisms yet. You must ensure the volumes are only used by one node at a time.
* In certain situations, such as draining a node, the volumes may be used simultaneously by two nodes. Before draining the node, first ensure the pods that use these volumes are deleted.
* Claims use the same conventions as volumes when requesting storage with specific access modes.

| Table 1. Supported Access Modes for Persistent Volumes | | | |
| --- | --- | --- | --- |
| Volume Plugin | ReadWriteOnce | ReadOnlyMany | ReadWriteMany |
| Fiber Channel | X | X | - |
| GlusterFS | X | X | X |
| iSCSI | X | X | - |
| NFS | X | X | X |

Reclaim Policy

The current reclaim policies are:

| Reclaim Policy | Description |
| --- | --- |
| Retain | Manual reclamation |
| Recycle | Basic scrub (e.g, rm -rf /<volume>/\*) |

Note: Currently, only NFS and HostPath support the 'Recycle' reclaim policy.

Different Phases:

A volumes can be found in one of the following phases:

| Phase | Description |
| --- | --- |
| Available | A free resource that is not yet bound to a claim. |
| Bound | The volume is bound to a claim. |
| Released | The claim has been deleted, but the resource is not yet reclaimed by the cluster. |
| Failed | The volume has failed its automatic reclamation. |

Note: The CLI shows the name of the PVC bound to the PV.

**Resources**

* Claims, like pods, can request specific quantities of a resource. In this case, the request is for storage. The same [resource model](https://github.com/kubernetes/community/blob/master/contributors/design-proposals/scheduling/resources.md#the-resource-model) applies to both volumes and claims.

PV

PVC

Pod

**Claims As Volumes**

* Pods access storage by using the claim as a volume.
* Claims must exist in the same namespace as the pod using the claim.
* The cluster finds the claim in the pod’s namespace and uses it to get the PersistentVolume backing the claim. The volume is then mounted to the host and into the pod:

kind: Pod

apiVersion: v1

metadata:

name: mypod

spec:

containers:

- name: myfrontend

image: dockerfile/nginx

volumeMounts:

- mountPath: "/var/www/html"

name: mypd

volumes:

- name: mypd

persistentVolumeClaim:

claimName: myclaim