Pv and Pvc

Introduction

Managing storage is a distinct problem from managing compute instances. The PersistentVolume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed. To do this, we introduce two new API resources: PersistentVolume and PersistentVolumeClaim.

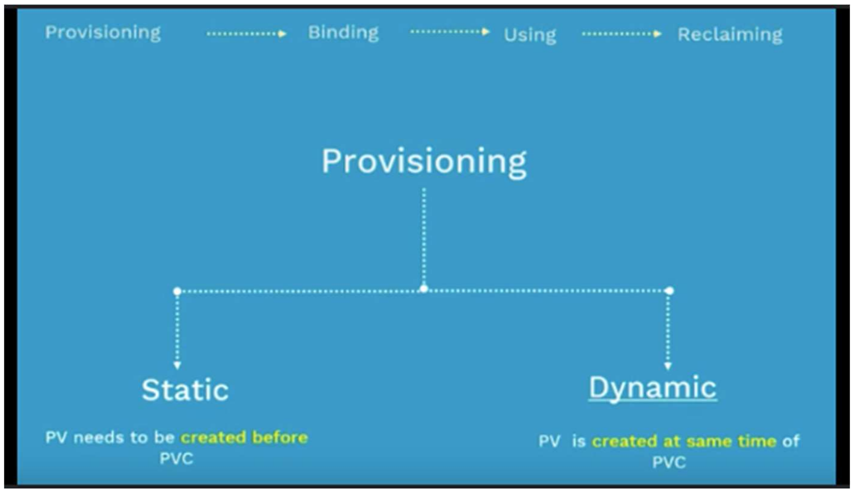
A PersistentVolume

- **(PV)** is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource.
- PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

A PersistentVolumeClaim

- **(PVC)** is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory).
- Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany, ReadWriteMany, or ReadWriteOncePod, see AccessModes).

There are two ways PVs may be provisioned: statically or dynamically.


Provisioning
Binding
Using
Reclaiming
Provisioning
Static
PV needs to be created before PVC
Dynamic
PV is created at same time of PVC

Persistent Volumes (PV)

Definition: Persistent Volumes are storage resources provisioned in a Kubernetes cluster, managed by the cluster administrator. They provide an abstraction layer over physical storage devices, allowing users to access storage without needing to know the underlying infrastructure details.

Available Options:

- **Capacity**: Specifies the size of the volume.
- **Access Modes:** Defines how the volume can be accessed. Below are the available options.
  1. ***ReadWriteOnce***: Can be mounted as read-write by a single node.
  2. ***ReadOnlyMany***: Can be mounted as read-only by multiple nodes.
  3. ***ReadWriteMany***: Can be mounted as read-write by multiple nodes.
- **Storage Class Name:** Associates the PV with a specific Storage Class.
- **Volume Mode:** Determines whether the volume is mounted as a filesystem or block device.

- **Persistent Volume Reclaim Policy:** Determines what happens to the PV's resources when released.
- **Mount Options:** Additional options to be passed to the mount command.

Manual PV Creation

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: example-pv
spec:
capacity:
storage: 10Gi
accessModes:
- ReadWriteOnce
storageClassName: manual
persistentVolumeReclaimPolicy: Retain
```

Persistent Volume Claims (PVC)

Definition: Persistent Volume Claims are requests for storage made by users or applications running in the cluster. They allow users to consume storage without needing to know the specifics of how it is provisioned.

Available Options:

- **Capacity**: Specifies the minimum size of the volume.
- **Access Modes:** Requests the required access mode.

- 1. ***ReadWriteOnce***: Can be mounted as read-write by a single node.
  2. ***ReadOnlyMany***: Can be mounted as read-only by multiple nodes.
  3. ***ReadWriteMany***: Can be mounted as read-write by multiple nodes.
- **Storage Class Name:** Requests a specific Storage Class.
- **Volume Mode:** Determines whether the volume is mounted as a filesystem or block device. Filesystem is the default mode used when volumeMode parameter is omitted.

Dynamic PV Provisioning with SC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: example-pvc
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 5Gi
storageClassName: standard
volumeMode: Filesystem
```

3. Storage Classes (SC)

Definition: Storage Classes define the types of storage available in the cluster and how they are provisioned. They enable dynamic provisioning of PVs based on predefined templates or policies.

Available Options:

- **Provisioner**: Specifies the type of volume plugin used for provisioning.
- **Parameters**: Custom parameters for the provisioner.
- **Reclaim Policy:** Determines what happens to the PV's resources when released. Below are the available options.

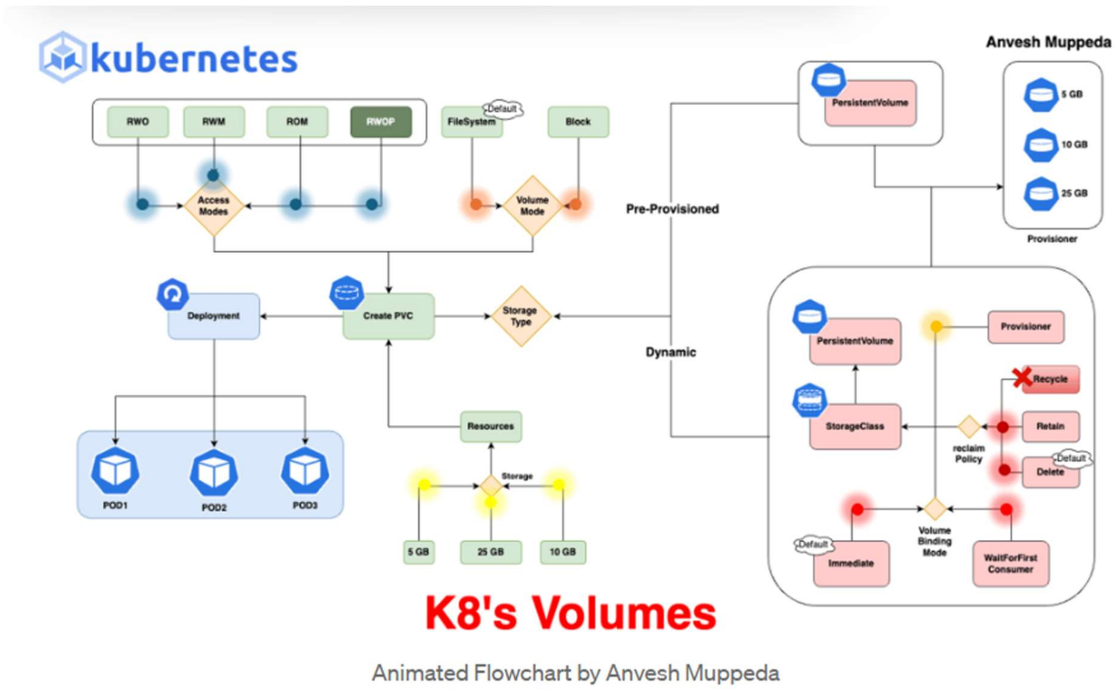
1. ***Retain*****:** When a PersistentVolumeClaim with the Retain policy is deleted, the associated PersistentVolume is released but retains data. Administrators manually reclaim the volume by deleting it and its associated data.
2. ***Delete*****:** For volume plugins that support the `Delete` reclaim policy, deletion removes both the PersistentVolume object from Kubernetes, as well as the associated storage asset in the external infrastructure.
3. ***Recycle***: `Recycle` reclaim policy performs a basic scrub (`rm -rf /thevolume/*`) on the volume and makes it available again for a new claim. (The `Recycle` reclaim policy is deprecated. Instead, the recommended approach is to use dynamic provisioning.)

- **Volume Binding Mode:** Determines when a PV is bound to a PVC. Below are the available options.

1. ***Immediate*** mode indicates that volume binding and dynamic provisioning occurs once the PersistentVolumeClaim is created.
2. ***WaitForFirstConsumer*** mode which will delay the binding and provisioning of a PersistentVolume until a Pod using the PersistentVolumeClaim is created.

Creating a Storage Class

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
type: gp2
reclaimPolicy: Delete
volumeBindingMode: Immediate
```


kubernetes
Anvesh Muppeda
RWO
RWM
ROM
RWOP
FileSystem
Default
Block
Access Modes
Volume Mode
Pre-Provisioned
PersistentVolume
5 GB
10 GB
25 GB
Provisioner
Deployment
Create PVC
Storage Type
Dynamic
POD1
POD2
POD3
Resources
Storage
5 GB
25 GB
10 GB
PersistentVolume
Provisioner
Recycle
StorageClass
reclaim Policy
Retain
Delete
Default
Default
Immediate
Volume Binding Mode
WaitForFirst Consumer
K8's Volumes
Animated Flowchart by Anvesh Muppeda