Persistent Volume (PV) & Persistent Volume Claim (PVC)

Introduction:

By default, Pods are having ephemeral storage so as soon as the Pod is terminated, all the data will be lost. So, Kubernetes introduces two API resources i.e Persistent Volume and Persistent Volume Claim.

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 (ReadWriteOnce, ReadOnlyMany, ReadWriteMany or ReadWriteOncePod)

Objectives:

1. Static Provisioning
2. Dynamic Provisioning

1. Manual Provisioning:

In this exercise, we are going to create a PV which will be referring to a particular storage or directory. This is a static process, so Administrator is responsible for providing the storage or creating the multiple PVs.

Let's do this exercise step by step.

Step 1: Create a directory:

We will access both the worker nodes and will create a directory which will be used by our PV. We are also creating an index.html file in that directory.

On worker1 node:

```
mkdir /mnt/data
```

```
root@worker1:~#
root@worker1:~# mkdir /mnt/data
root@worker1:~# echo "hello from worker node1" > /mnt/data/index.html
root@worker1:~#
```

On worker2 node:

```
mkdir /mnt/data
```

```
root@worker2:~#
root@worker2:~# mkdir /mnt/data
root@worker2:~#
root@worker2:~# echo "hello from worker node2" > /mnt/data/index.html
root@worker2:~#
```

Above we have created a directory **/mnt/data** on each worker node which will be used by our PV.

Step2: Create a Persistent Volume:

Now we are going to create a persistent volume using the below Yaml file.

We get back to our master node and will create a Yaml file.

```
vi pv.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
      path: "/mnt/data"
```

Save the above file and apply using the below command. Under hostPath, we are mentioning the same directory which we have created on worker machines.

```
kubectl apply -f pv.yaml
```

```
root@master:~# kubectl apply -f pv.yaml
persistentvolume/pv-volume created
root@master:~#
```

```
root@master:~# kubectl get pv
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
pv-volume   10Gi       RWO            Retain           Available           manual                  77s
root@master:~#
```

```
root@master:~# kubectl describe pv pv-volume
Name:            pv-volume
Labels:          type=local
Annotations:     <none>
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    manual
Status:          Available
Claim:
Reclaim Policy:  Retain
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        10Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /mnt/data
    HostPathType:
Events:            <none>
root@master:~#
```

From above output we can see that the status of this PV is available and using the default StorageClass which is manual. As mentioned in the description, the Reclaim Policy is Retain which is default one.

There are three types of Reclaim Policies which PVs are using.

1. Retain: The Retain reclaim policy allows for manual reclamation of the resource. An administrator will delete the persistent volume, clean the data and delete the storage asset.
2. Delete: This policy removes both the PersistentVolume object from Kubernetes, as well as the associated storage asset in the external infrastructure, such as an AWS EBS, GCE PD, Azure Disk, or Cinder volume.
3. Recycle: the Recycle reclaim policy performs a basic scrub (rm -rf /thevolume/*) on the volume and makes it available again for a new claim.

Step 3: Create a Persistent Volume Claim:

Now we need to create a PVC which will be consuming the available PV. The access mode should match and request should be equal or less than the storage configured in the PV. Use the below Yaml file to create a PVC.

```
vi pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Above we are using a manual storage class so it will look for a PV in our cluster. Access mode is ReadWriteOnce which should match with our PV and storage is 3Gi which is less that the storage of our PV. Let's apply the definition file.

```
Kubectl apply -f pvc.yaml
```

```
root@master:~# kubectl apply -f pvc.yaml
persistentvolumeclaim/task-pv-claim created
root@master:~#
root@master:~# kubectl get pvc
NAME            STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Bound    pv-volume   10Gi       RWO            manual         7s
root@master:~#
```

```
root@master:~#
root@master:~# kubectl describe pvc task-pv-claim
Name:          task-pv-claim
Namespace:     default
StorageClass:  manual
Status:        Bound
Volume:        pv-volume
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      10Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Used By:       <none>
Events:        <none>
root@master:~#
root@master:~#
```

As soon as we apply the pvc definition file, it comes to bound status with volume pv-volume as seen in above output.

If we see the PV status as well, it will also show that the status is bound with PVC task-pv-claim.

```
root@master:~#
root@master:~#
root@master:~# kubectl get pv
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
pv-volume   10Gi       RWO            Retain           Bound    default/task-pv-claim   manual                  25m
root@master:~#
```

```
root@master:~# kubectl describe pv pv-volume
Name:            pv-volume
Labels:          type=local
Annotations:     pv.kubernetes.io/bound-by-controller: yes
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    manual
Status:          Bound
Claim:           default/task-pv-claim
Reclaim Policy:  Retain
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        10Gi
Node Affinity:   <none>
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /mnt/data
    HostPathType:
Events:            <none>
root@master:~#
```

Step 4: Create a Pod:

Now are going to create a Pod which will be referring the PVC. Use the below Yaml file.

vi pv-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

In above definition file, we are referring to persistent volume claim which is task-pv-claim in our case. Now let's apply the changes and see the result.

```
kubectl apply -f pv-pod.yaml
```

```
root@master:~#
root@master:~# kubectl get pods -o wide
NAME     READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
pv-pod   1/1     Running   0          31s   192.168.235.179   worker1   <none>           <none>
root@master:~#

root@master:~#
root@master:~# curl 192.168.235.179
hello from worker node1
root@master:~#
```

We can see that the pod has been deployed on worker1 machine and if we curl it, we can see the desired result. It is accessing the file stored in our worker1 machine.

2. Dynamic Provisioning

Now this method uses a storage class, we need not to create a PV. We will create a PVC which will referring to the storage class. The storage class will take care of dynamic PV creation. Let's proceed with this exercise.

Step 1: Check the Storage Class

We are using EKS cluster to use dynamic provisioning. AWS provides gp2 Storage class. Check the details below using the commands.

```
kubectl get sc
```

```
kubectl describe sc gp2
```

```
root@ip-172-31-33-180:~# kubectl get sc
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gp2 (default)   kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false                  85m
```

```
root@ip-172-31-33-180:~# kubectl describe sc gp2
Name:            gp2
IsDefaultClass:  Yes
Annotations:     kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClass","metadata":{"annotations":{"storagec
lass.kubernetes.io/is-default-class":"true"},"name":"gp2"},"parameters":{"fsType":"ext4","type":"gp2"},"provisioner":"kubernetes.io/aws-ebs","volumeBindingMod
e":"WaitForFirstConsumer"}
,storageclass.kubernetes.io/is-default-class=true
Provisioner:           kubernetes.io/aws-ebs
Parameters:            fsType=ext4,type=gp2
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
root@ip-172-31-33-180:~#
```

Step 2: Create a PVC:

Use the below Yaml file to create a PVC.

vi pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: gp2
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
       storage: 3Gi
```

In above file, we have mentioned the storageClassName as gp2. Now let's apply this file.

kubectl apply -f pvc.yaml

```
root@ip-172-31-33-180:~# vi pvc.yaml
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl apply -f pvc.yaml
persistentvolumeclaim/task-pv-claim created
root@ip-172-31-33-180:~#
```

```
root@ip-172-31-33-180:~# kubectl get pvc
NAME           STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Pending                                      gp2            8s
root@ip-172-31-33-180:~#
```

```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl get pv
No resources found
root@ip-172-31-33-180:~#
```

As we can see that the PVC is still in pending state because the storage class **VolumeBindingMode** is **waitForFirstConsumer.** As soon as a Pod is created referring to this PVC, only then a PV will be created dynamically.

Step 2: Create a Pod

Now lets create a pod which will be referring to our PVC. Use the below Yaml file to create a Pod.

vi pv-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Above Pod definition file is referring to task-pv-claim, let's apply this file.

```
kubectl apply -f pv-pod.yaml
```

As soon as we created this pod, our PVC will come to bound state and storage class will create a PV dynamically.

```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# vi pv-pod.yaml
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl apply -f pv-pod.yaml
pod/task-pv-pod created
root@ip-172-31-33-180:~#
```

```
root@ip-172-31-33-180:~# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
task-pv-pod   1/1     Running   0          42s
root@ip-172-31-33-180:~#
```

```
root@ip-172-31-33-180:~# kubectl get pvc
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Bound    pvc-d79dd321-aaea-4384-af18-b54c9dc4107b   3Gi        RWO            gp2            2m11s
root@ip-172-31-33-180:~#
```

```
root@ip-172-31-33-180:~# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
pvc-d79dd321-aaea-4384-af18-b54c9dc4107b   3Gi        RWO            Delete           Bound    default/task-pv-claim   gp2                     56s
```

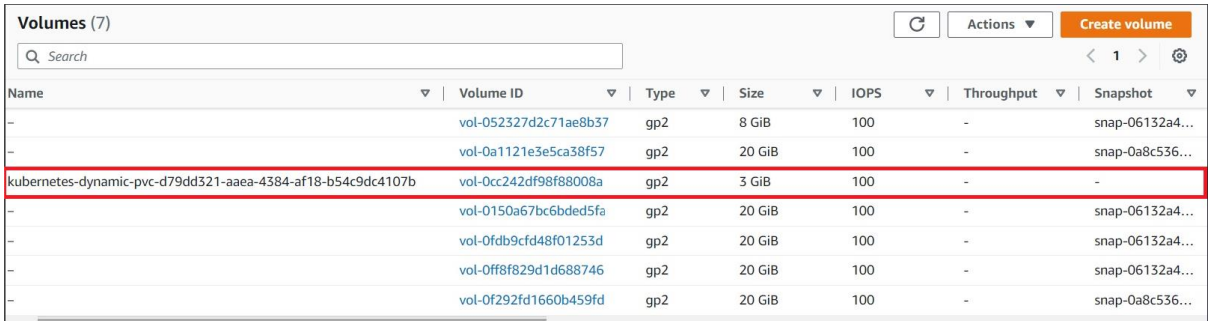
Now the output shows that PV has been created and it is bound to our PVC. We can further get more details about the PVC and PV using the describe command.

```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl describe pvc task-pv-claim
Name:          task-pv-claim
Namespace:     default
StorageClass:  gp2
Status:        Bound
Volume:        pvc-d79dd321-aaea-4384-af18-b54c9dc4107b
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
               pv.kubernetes.io/bound-by-controller: yes
               volume.beta.kubernetes.io/storage-provisioner: kubernetes.io/aws-ebs
               volume.kubernetes.io/selected-node: ip-172-31-18-184.us-east-2.compute.internal
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      3Gi
Access Modes:  RWO
VolumeMode:    Filesystem
Used By:       task-pv-pod
Events:
  Type    Reason                Age                   From                         Message
  ----    ------                ----                  ----                         -------
  Normal  WaitForFirstConsumer  2m26s (x6 over 3m29s)  persistentvolume-controller  waiting for first consumer to be created before binding
  Normal  ProvisioningSucceeded  2m7s                  persistentvolume-controller  Successfully provisioned volume pvc-d79dd321-aaea-4384-af18-b54c9dc4107b
using kubernetes.io/aws-ebs
root@ip-172-31-33-180:~#
```

Below we can see the PV's source details which is AWS in our case and bound to the claim **task-pv-claim**.

```
root@ip-172-31-33-180:~#
root@ip-172-31-33-180:~# kubectl describe pv pvc-d79dd321-aaea-4384-af18-b54c9dc4107b
Name:              pvc-d79dd321-aaea-4384-af18-b54c9dc4107b
Labels:            topology.kubernetes.io/region=us-east-2
                   topology.kubernetes.io/zone=us-east-2b
Annotations:       kubernetes.io/createdby: aws-ebs-dynamic-provisioner
                   pv.kubernetes.io/bound-by-controller: yes
                   pv.kubernetes.io/provisioned-by: kubernetes.io/aws-ebs
Finalizers:        [kubernetes.io/pv-protection]
StorageClass:      gp2
Status:            Bound
Claim:             default/task-pv-claim
Reclaim Policy:    Delete
Access Modes:      RWO
VolumeMode:        Filesystem
Capacity:          3Gi
Node Affinity:
  Required Terms:
    Term 0:        topology.kubernetes.io/zone in [us-east-2b]
                   topology.kubernetes.io/region in [us-east-2]
Message:
Source:
    Type:        AWSElasticBlockStore (a Persistent Disk resource in AWS)
    VolumeID:    aws://us-east-2b/vol-0cc242df98f88008a
    FSType:      ext4
    Partition:   0
    ReadOnly:    false
Events:          <none>
root@ip-172-31-33-180:~#
```

Below snapshot shows that a physical disk has been created dynamically as soon as we create a pod.


Volumes (7)
Actions
Create volume
Search
1
Name
Volume ID
Type
Size
IOPS
Throughput
Snapshot
– vol-052327d2c71ae8b37 gp2 8 GiB 100 - snap-06132a4...
– vol-0a1121e3e5ca38f57 gp2 20 GiB 100 - snap-0a8c536...
kubernetes-dynamic-pvc-d79dd321-aaea-4384-af18-b54c9dc4107b vol-0cc242df98f88008a gp2 3 GiB 100 - -
– vol-0150a67bc6bded5fa gp2 20 GiB 100 - snap-06132a4...
– vol-0fdb9cfd48f01253d gp2 20 GiB 100 - snap-06132a4...
– vol-0ff8f829d1d688746 gp2 20 GiB 100 - snap-06132a4...
– vol-0f292fd1660b459fd gp2 20 GiB 100 - snap-0a8c536...