KUBERNETES PERSISTENT STORAGE

Intree-Plugins: Earlier, storage in Kubernetes was managed through in-tree plugins, but adding new features required updates to the Kubernetes core, making enhancements slow. Therefore, in-tree plugins have been deprecated in favor of more flexible external solutions.

Ex: awsElasticBlockStore (AWS EBS)

azureDisk, azureFile

gcePersistentDisk (GCE PD) etc

CSI Drivers: Kubernetes has decoupled storage management by adopting CSI plugins, the modern approach for integrating external storage. Storage is now a non-core component, allowing vendors to develop plugins independently, enabling faster innovation and advancements.

Ex: AWS EBS CSI Driver: ebs.csi.aws.com

GCE PD CSI Driver: pd.csi.storage.gke.io

Azure Disk CSI Driver: disk.csi.azure.com

HostPath: HostPath is an in-tree plugin used to mount a file or directory from the host node's filesystem directly into a pod. It behaves like a persistent volume since the data remains independent of the pod lifecycle. However, Kubernetes does not recommend using HostPath in production environments due to security and reliability concerns.

Use Cases: Debugging, Accessing host-level files (logs, configuration files), or sharing specific host resources with containers.

Lets Explore the Concept of HostPath practically!

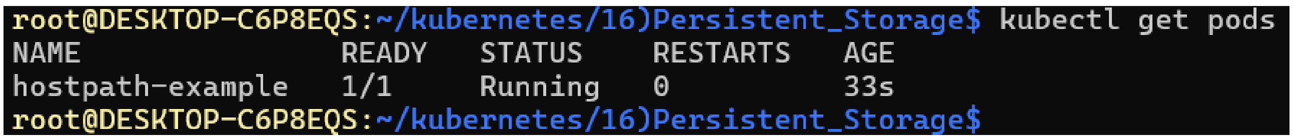
Hostpath.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-example
```

```
spec:
  volumes:
    - name: host-volume
      hostPath:
        path: /tmp/hostfile
        type: FileOrCreate
  containers:
    - name: busybox-container
      image: busybox
      command: ["/bin/sh", "-c", "cat /data &&
sleep 3600"]
      volumeMounts:
        - name: host-volume
          mountPath: /data
```

Apply;

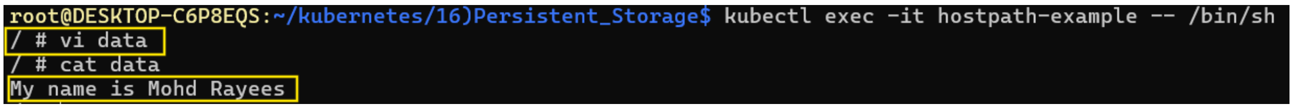
```
kubectl apply -f hostpath.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pods
NAME               READY   STATUS    RESTARTS   AGE
hostpath-example   1/1     Running   0          33s
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$
```

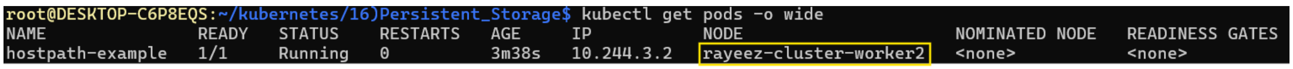
Exec into the pod and write some data into /data file:

```
kubectl exec -it hostpath-example -- /bin/sh
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl exec -it hostpath-example -- /bin/sh
/ # vi data
/ # cat data
My name is Mohd Rayees
```

```
kubectl get pods -o wide
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pods -o wide
NAME               READY   STATUS    RESTARTS   AGE     IP           NODE                     NOMINATED NODE   READINESS GATES
hostpath-example   1/1     Running   0          3m38s   10.244.3.2   rayeez-cluster-worker2   <none>           <none>
```

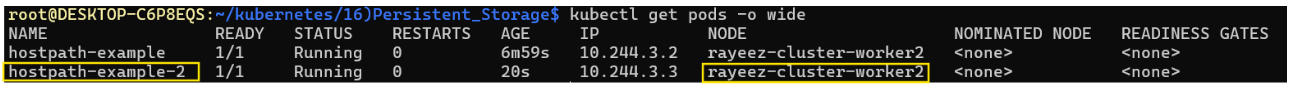
Pod is running on Worker Node 2. Let Schedule one more pod on the same Node.

Hostpath-2.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-example-2
spec:
  nodeName: rayeez-cluster-worker2
  volumes:
    - name: host-volume
      hostPath:
        path: /tmp/hostfile
        type: FileOrCreate
  containers:
    - name: busybox-container
      image: busybox
      command: ["/bin/sh", "-c", "cat /data &&
sleep 3600"]
      volumeMounts:
        - name: host-volume
          mountPath: /data
```

Apply,

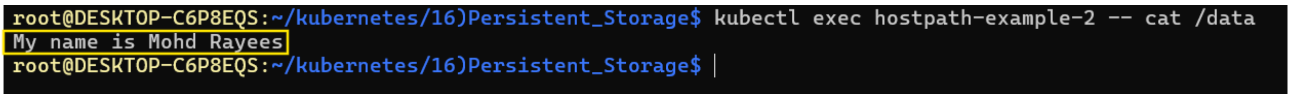
```
kubectl apply -f hostpath-2.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pods -o wide
NAME                 READY   STATUS    RESTARTS   AGE     IP           NODE                     NOMINATED NODE   READINESS GATES
hostpath-example     1/1     Running   0          6m59s   10.244.3.2   rayeez-cluster-worker2   <none>           <none>
hostpath-example-2   1/1     Running   0          20s     10.244.3.3   rayeez-cluster-worker2   <none>           <none>
```

Exec into pod 2 and try to access the /data file:

```
kubectl exec hostpath-example-2 -- cat /data
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl exec hostpath-example-2 -- cat /data
My name is Mohd Rayees
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ |
```

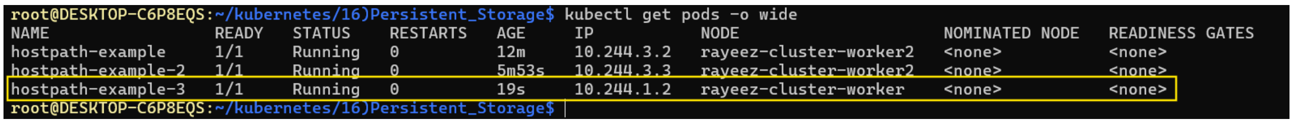
Lets schedule a pod on Worker node-1 deliberately and look at the container's /data file:

Hostpath-3.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-example-3
spec:
  nodeName: rayeez-cluster-worker
  volumes:
    - name: host-volume
      hostPath:
          path: /tmp/hostfile
          type: FileOrCreate
  containers:
    - name: busybox-container
      image: busybox
      command: ["/bin/sh", "-c", "cat /data &&
sleep 3600"]
      volumeMounts:
        - name: host-volume
          mountPath: /data
```

Apply;

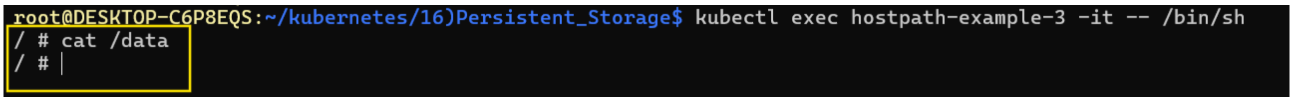
```
kubectl apply -f hostpath-3.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pods -o wide
NAME                 READY   STATUS    RESTARTS   AGE     IP           NODE                    NOMINATED NODE   READINESS GATES
hostpath-example     1/1     Running   0          12m     10.244.3.2   rayeez-cluster-worker2   <none>           <none>
hostpath-example-2   1/1     Running   0          5m53s   10.244.3.3   rayeez-cluster-worker2   <none>           <none>
hostpath-example-3   1/1     Running   0          19s     10.244.1.2   rayeez-cluster-worker    <none>           <none>
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$
```

Exec into pod and access the /data:

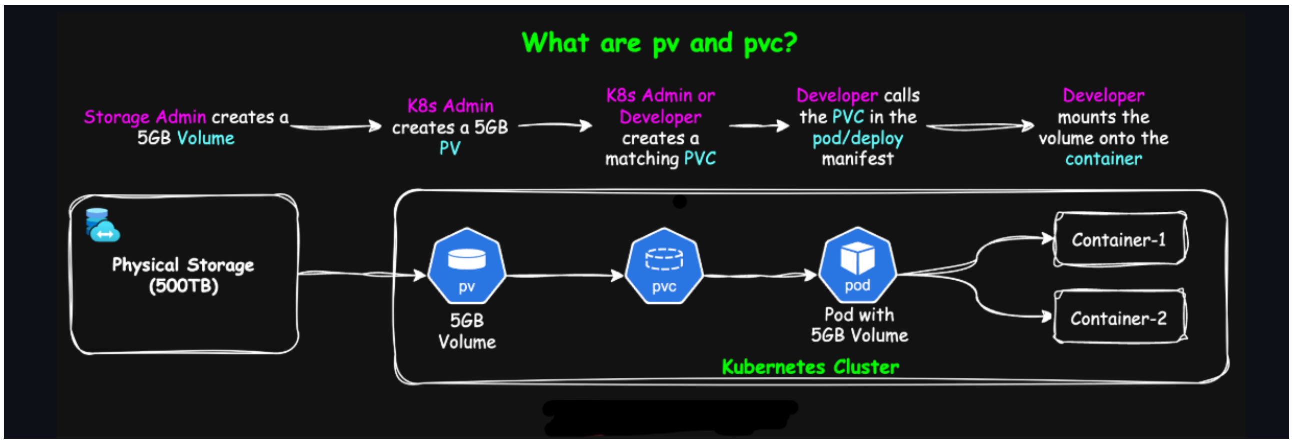
```
kubectl exec hostpath-example-3 -it -- /bin/sh
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl exec hostpath-example-3 -it -- /bin/sh
/ # cat /data
/ #
```

HostPath Storage is shared between containers inside a Pod and between various pods running on the same Node. Pods running on node B can't access HostPath storage of Node A.

Understanding Concepts of PV and PVC in Kubernetes:


What are pv and pvc?
Storage Admin creates a 5GB Volume
K8s Admin creates a 5GB PV
K8s Admin or Developer creates a matching PVC
Developer calls the PVC in the pod/deploy manifest
Developer mounts the volume onto the container
Physical Storage (500TB)
pv
5GB Volume
pvc
pod
Pod with 5GB Volume
Container-1
Container-2
Kubernetes Cluster

MOHD RAYEES

Persistent Volumes (PVs):

- A PV is a piece of storage in your cluster. Its Cluster scoped.
- PVs exist independently of Pod lifecycles and can be reused or retained even after the Pod is deleted
- It has properties such as **capacity, access modes, and reclaim policies**

Persistent Volume Claims (PVCs):

- A PVC is a request for storage by a user. Its Namespaced.
- It functions similarly to how a Pod requests compute resources
- When a PVC is created, Kubernetes searches for a PV that meets the claim's requirements

PV and PVC are bound to each other in **strict one-to-one relationship.**

Access Modes in Kubernetes Persistent Volumes:

ReadWriteOnce (RWO): Storage is mounted by a single node with read, write permissions. It can be accessed by multiple pods that are running on the name node. Block Storage is used typically.

ReadOnlyMany (ROX): Storage is mounted by multiple nodes with Read only permissions. Multiple pods running on various nodes of cluster can access this storage. File Storage is used typically.

ReadWriteMany (RWX): Storage is mounted by multiple Nodes with read, write permissions. Multiple pods running on various nodes of cluster can access this storage. File Storage is used typically.

ReadWriteOncePod (RWOP): Storage is mounted by a single node with read, write permissions. It is accessed by single pod across the cluster. Block Storage is used typically.

Reclaim Policies in Kubernetes:

1)Delete: When a PVC is deleted, Its bounded PV also get deleted , clearing the storage. Its useful in Cloud Environments to save Cloud Costs.

2)Retain: Even if a PVC is deleted, PV remains in the cluster and moves to Released state. This PV can be rebind to another PVC editing its claimref. It can also used for auditing, compliance and data recovery.

Must Matching Conditions for PVC and PV to bound:

Condition	PVC Requirement	PV Must Have
Storage Capacity	10 GB	>=10GB
Access Mode	ReadWriteMany	ReadWriteMany
Storage Class	fast-ssd	fast-ssd
Volume State	Unbound	Available
Volume Mode	Filesystem/Block	Filesystem/Block

Lets Explore PV and PVC practically!

Note: Iam using a KIND cluster. It comes with a default storageclass “standard”. I took its back up in sc.yaml, deleted it and Performed the demo manually.

Pv.yaml:

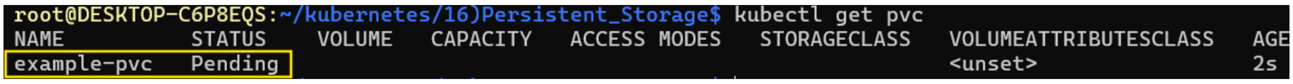
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /mnt/data
```

pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

Apply each yaml manifest one by one:

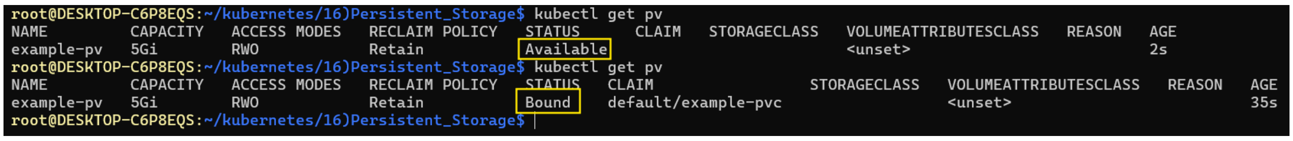
```
kubectl apply -f pvc.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pvc
NAME          STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
example-pvc   Pending                                                     <unset>                 2s
```

The PVC is in a Pending state because it hasn't found a matching Persistent Volume yet.

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

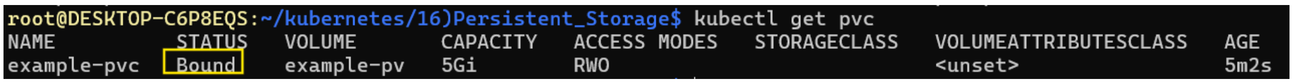


```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
example-pv   5Gi        RWO            Retain           Available                          <unset>                          2s
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
example-pv   5Gi        RWO            Retain           Bound    default/example-pvc                  <unset>                          35s
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$
```

The PV was initially in the Available state, and once it found a matching PVC, its status changed to Bound.

```
kubectl get pvc
```

Similarly, PVC status is changed to bound after PV is created:



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pvc
NAME          STATUS   VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
example-pvc   Bound    example-pv   5Gi        RWO                           <unset>                 5m2s
```

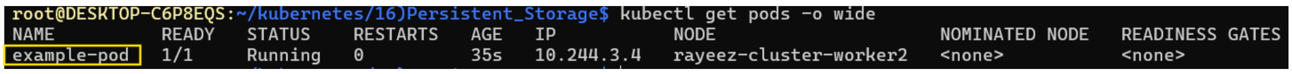
Now create a Pod that uses this PVC.:

pod.yaml:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
      volumeMounts:
        - name: persistent-storage
          mountPath: /usr/share/nginx/html
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: example-pvc
```

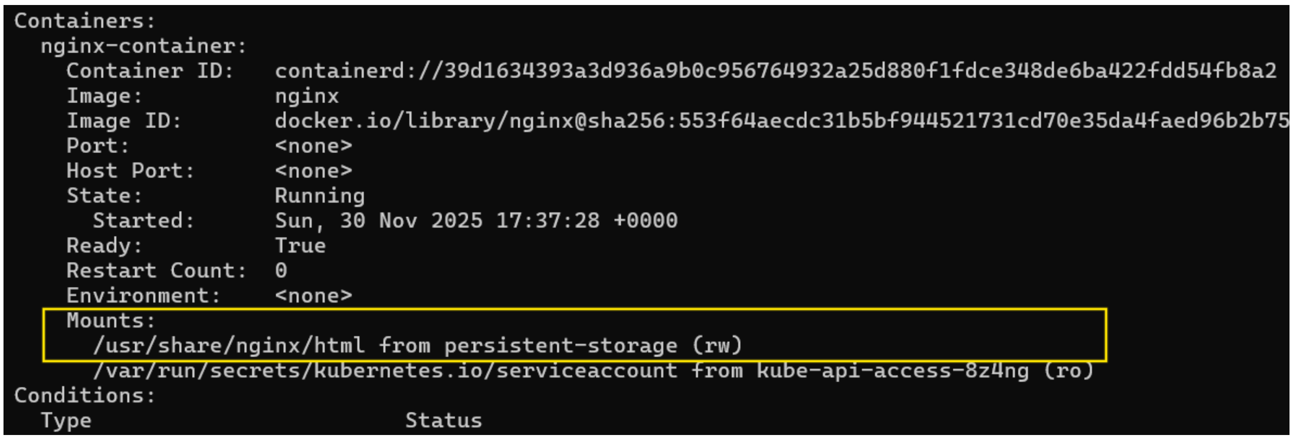
Apply;

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

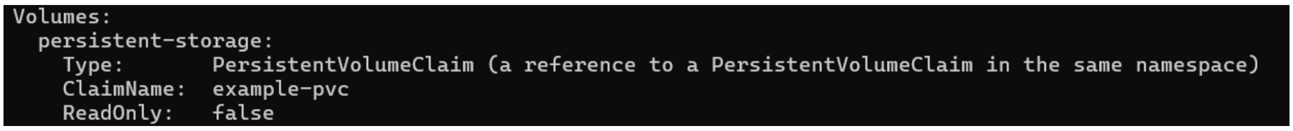


```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE                    NOMINATED NODE   READINESS GATES
example-pod   1/1     Running   0          35s   10.244.3.4   rayeez-cluster-worker2  <none>           <none>
```

```
kubectl describe pods example-pod
```



```
Containers:
  nginx-container:
    Container ID:   containerd://39d1634393a3d936a9b0c956764932a25d880f1fdce348de6ba422fdd54fb8a2
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:553f64aecdc31b5bf944521731cd70e35da4faed96b2b75
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sun, 30 Nov 2025 17:37:28 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /usr/share/nginx/html from persistent-storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-8z4ng (ro)
Conditions:
  Type                          Status
```



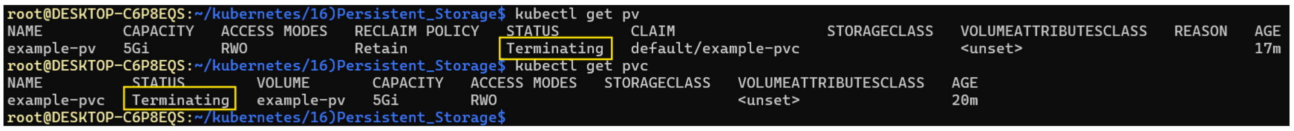
```
Volumes:
  persistent-storage:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  example-pvc
    ReadOnly:   false
```

MOHD RAYEES

The persistent storage volume is now mounted at the specified path inside the container, as shown.

Lets delete both PV and PVC and see their status:

```
kubectl delete pvc example-pvc
kubectl delete pv example-pv
```



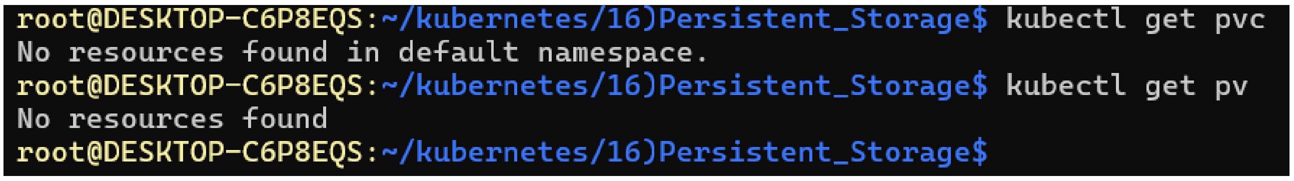
```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS        CLAIM                 STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
example-pv   5Gi        RWO            Retain           Terminating   default/example-pvc                  <unset>                          17m
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pvc
NAME          STATUS        VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
example-pvc   Terminating   example-pv   5Gi        RWO                           <unset>                 20m
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$
```

PV and PVC are in Terminating state, But not terminated.

This is a Kubernetes protection mechanism: a PV is deleted only when its PVC is deleted, and the PVC is deleted only when the Pod using it is removed.

Lets delete the pod and again check PV, PVC status:

```
kubectl delete pods example-pod
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pvc
No resources found in default namespace.
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pv
No resources found
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$
```

Both PV and PVC are deleted now.

StorageClasses:

A Storage Class in Kubernetes is a way to define different storage configurations, enabling dynamic provisioning of Persistent Volumes (PVs).

It eliminates the need to manually pre-create PVs and provides flexibility for managing storage across diverse workloads.

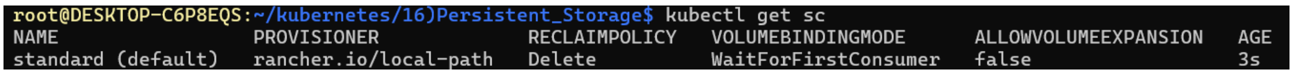
WaitForFirstConsumer: It is a volume binding mode that ensures that the volume is created only after the Pod is scheduled.

Lets apply the sc.yaml (the back up that we have taken earlier):

```
apiVersion: v1
items:
  - apiVersion: storage.k8s.io/v1
    kind: StorageClass
    metadata:
      storageclass.kubernetes.io/is-default-
class: "true"
      name: standard
    provisioner: rancher.io/local-path
    reclaimPolicy: Delete
    volumeBindingMode: WaitForFirstConsumer
```

Apply;

```
kubectl apply -f sc.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get sc
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
standard (default)   rancher.io/local-path   Delete          WaitForFirstConsumer   false                  3s
```

This StorageClass is set as the default, so even if a PVC does not explicitly reference it, Kubernetes will still use it to dynamically provision a PV.

pvc.yaml:

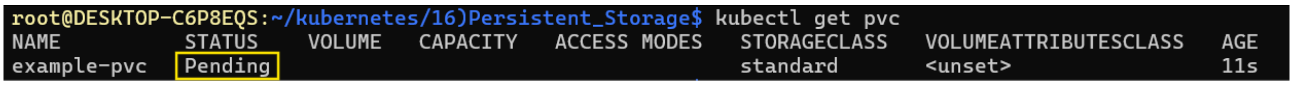
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

Apply;

```
kubectl apply -f pvc.yaml
```

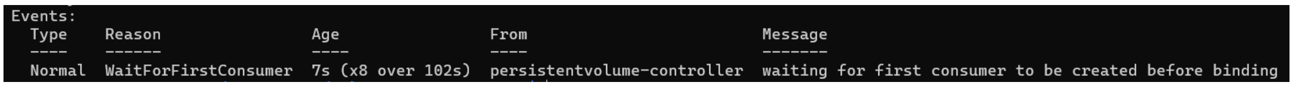
Verify;

```
kubectl get pvc
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pvc
NAME          STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
example-pvc   Pending                                      standard       <unset>                 11s
```

```
kubectl describe pvc example-pvc
```



```
Events:
  Type    Reason                Age               From                         Message
  ----    ------                ----              ----                         -------
  Normal  WaitForFirstConsumer  7s (x8 over 102s)  persistentvolume-controller  waiting for first consumer to be created before binding
```

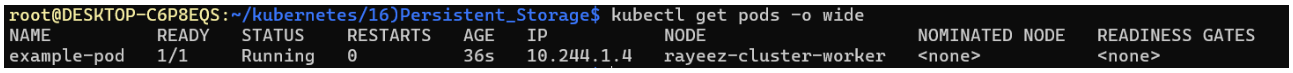
Because the StorageClass uses the **WaitForFirstConsumer** volume binding mode, the PV will not be created until a Pod that uses the PVC is created.

Lets create the pod using this PVC:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
      volumeMounts:
        - name: persistent-storage
          mountPath: /usr/share/nginx/html
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: example-pvc
```

Apply;

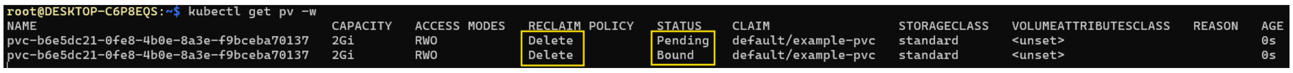
```
kubectl apply -f pod.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/16)Persistent_Storage$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE                    NOMINATED NODE   READINESS GATES
example-pod   1/1     Running   0          36s   10.244.1.4   rayeez-cluster-worker   <none>           <none>
```

```
kubectl get pv -w
```

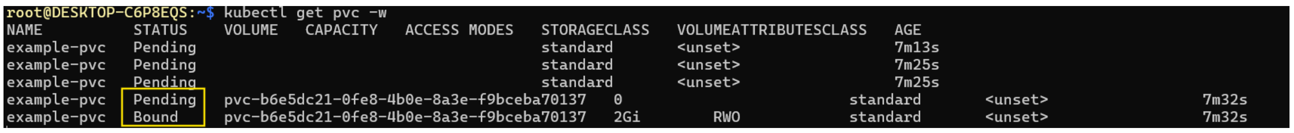
Once this pod referencing the PVC is created, PV also created by Storage Class dynamically:



```
root@DESKTOP-C6P8EQS:~$ kubectl get pv -w
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS    CLAIM                 STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pvc-b6e5dc21-0fe8-4b0e-8a3e-f9bceba70137   2Gi        RWO            Delete           Pending   default/example-pvc   standard       <unset>                          0s
pvc-b6e5dc21-0fe8-4b0e-8a3e-f9bceba70137   2Gi        RWO            Delete           Bound     default/example-pvc   standard       <unset>                          0s
```

Status of PV is changed from pending to bound because it has found its matching PVC.

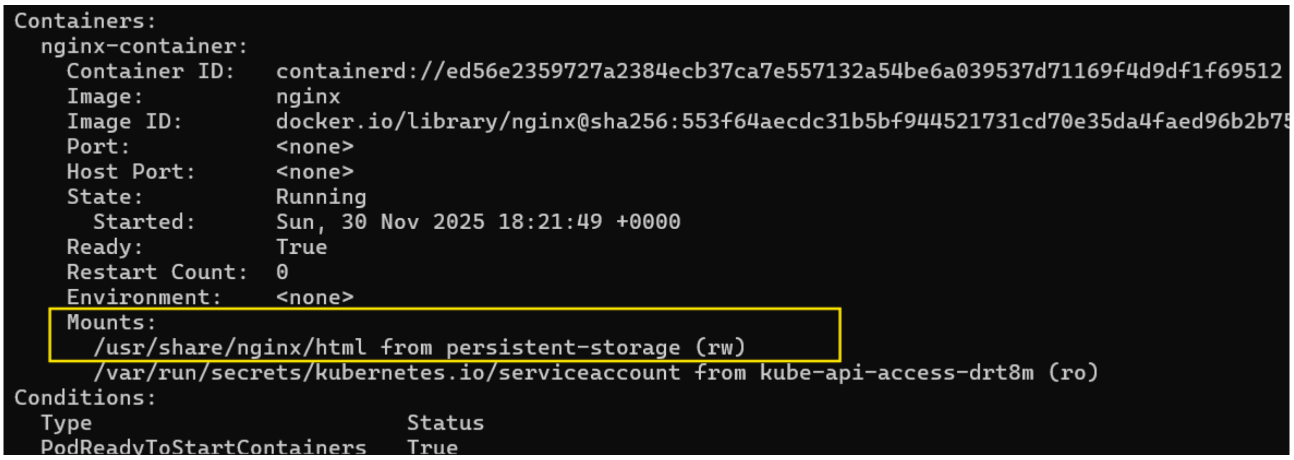
```
kubectl get pvc -w
```



```
root@DESKTOP-C6P8EQS:~$ kubectl get pvc -w
NAME          STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
example-pvc   Pending                                      standard       <unset>                 7m13s
example-pvc   Pending                                      standard       <unset>                 7m25s
example-pvc   Pending                                      standard       <unset>                 7m25s
example-pvc   Pending   pvc-b6e5dc21-0fe8-4b0e-8a3e-f9bceba70137   0                    standard       <unset>                 7m32s
example-pvc   Bound     pvc-b6e5dc21-0fe8-4b0e-8a3e-f9bceba70137   2Gi        RWO       standard       <unset>                 7m32s
```

The status of PVC is changed from pending to bound, because it has found its matching PV.

```
kubectl describe pods example-pod
```

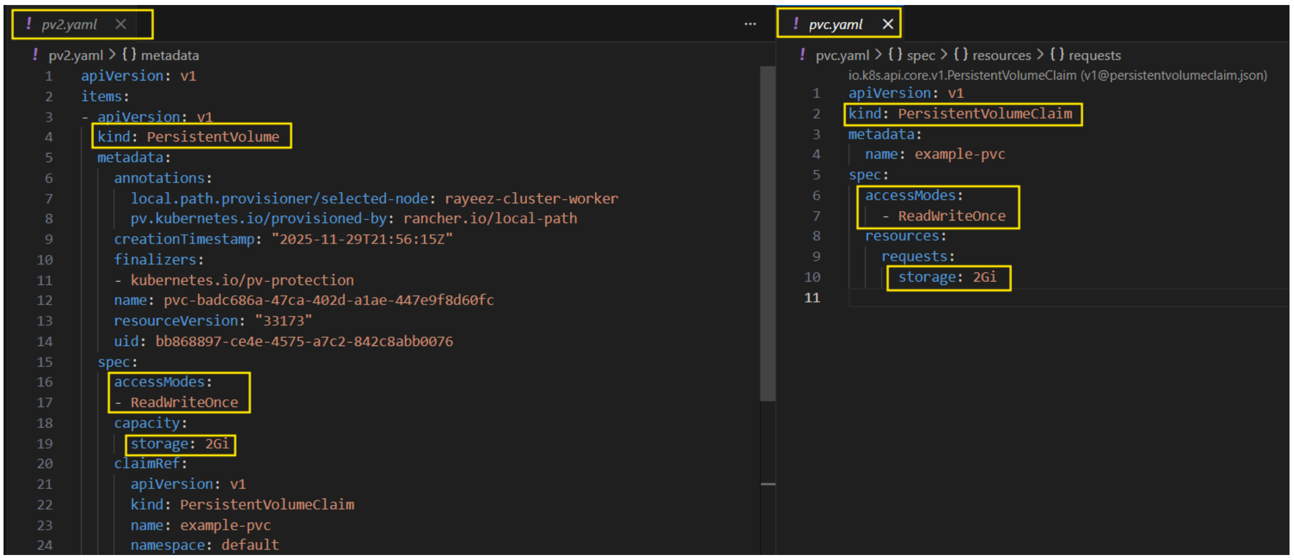


```
Containers:
  nginx-container:
    Container ID:   containerd://ed56e2359727a2384ecb37ca7e557132a54be6a039537d71169f4d9df1f69512
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:553f64aecdc31b5bf944521731cd70e35da4faed96b2b75
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sun, 30 Nov 2025 18:21:49 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /usr/share/nginx/html from persistent-storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-drt8m (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
```

Volume is mounted on the container as specified in pod manifest.

Let compare PVC and PV(created by SC) manifests:

```
Kubectl get pv -o yaml > pv2.yaml
```



pv2.yaml

pv2.yaml > { } metadata

```yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: PersistentVolume
  metadata:
    annotations:
      local.path.provisioner/selected-node: rayeez-cluster-worker
      pv.kubernetes.io/provisioned-by: rancher.io/local-path
    creationTimestamp: "2025-11-29T21:56:15Z"
    finalizers:
    - kubernetes.io/pv-protection
    name: pvc-badc686a-47ca-402d-a1ae-447e9f8d60fc
    resourceVersion: "33173"
    uid: bb868897-ce4e-4575-a7c2-842c8abb0076
  spec:
    accessModes:
    - ReadWriteOnce
    capacity:
      storage: 2Gi
    claimRef:
      apiVersion: v1
      kind: PersistentVolumeClaim
      name: example-pvc
      namespace: default
```

pvc.yaml

pvc.yaml > { } spec > { } resources > { } requests

io.k8s.api.core.v1.PersistentVolumeClaim (v1@persistentvolumeclaim.json)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi

```

Storage class has provision a PV matching with PVC requirements.