***Kubernetes Persistent Volume (PV), Persistent Volume Claim (PVC), and Deployment Example***

This documentation explains how to use **Persistent Volumes (PV)** and **Persistent Volume Claims (PVC)** in Kubernetes with a running example.

**1. What is a Persistent Volume (PV)?**

* A **Persistent Volume** is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using a StorageClass.
* PVs are cluster resources, just like nodes.

Key properties:

* Capacity (e.g., 1Gi, 5Gi)
* Access Modes (ReadWriteOnce, ReadOnlyMany, ReadWriteMany)
* Reclaim Policy (Retain, Recycle, Delete)

**2. What is a Persistent Volume Claim (PVC)?**

* A **Persistent Volume Claim** is a request for storage by a user.
* PVCs consume PVs.
* PVCs can request size, access modes, and a specific StorageClass.

**3. Example Setup**

**a) PersistentVolume (PV)**

apiVersion: v1

kind: PersistentVolume

metadata:

name: localpv

labels:

app: local

spec:

capacity:

storage: 1Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Retain

storageClassName: local-storage

hostPath: # works for local single-node clusters like Minikube/Kind

path: /mnt/data # Use a Linux path inside the node, not Windows path

⚠️ Note: hostPath works only for local clusters or testing. For production, use cloud volumes (EBS, Azure Disk, GCE Persistent Disk, NFS, etc.).

**b) PersistentVolumeClaim (PVC)**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: localpv

namespace: nginx

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

storageClassName: local-storage

* PVC requests 1Gi from the StorageClass local-storage.
* The PVC will bind to the localpv we created above.

**c) Deployment with PVC**

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

namespace: nginx

spec:

replicas: 1

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: nginx-pod

image: nginx:latest

ports:

- containerPort: 80

volumeMounts:

- mountPath: /usr/share/nginx/html # inside container

name: myvolume

volumes:

- name: myvolume

persistentVolumeClaim:

claimName: localpv

Here:

* Nginx serves files from /usr/share/nginx/html.
* That directory is backed by our PVC (localpv).
* Data inside /mnt/data on the host will be available inside the container.

**4. Commands to Deploy**

# Create namespace

kubectl create ns nginx

# Apply PV

kubectl apply -f storage.yml

# Apply PVC (in nginx namespace)

kubectl apply -f pvc.yml

# Apply Deployment

kubectl apply -f deployment.yml

# Describe pod to see if PVC is mounted

kubectl describe pod <pod-name> -n nginx

If everything works, the pod will mount the PVC, and Nginx will use the persistent volume for its data directory.

**Statefulset case:**  
  
**How volumeClaimTemplates work in StatefulSets**

* In a **Deployment**, you usually create a PVC separately, then mount it.
* In a **StatefulSet**, you don’t manually create PVCs. Instead, you declare a volumeClaimTemplates.

👉 Kubernetes will **auto-create a PVC per replica** of the StatefulSet.

Example:  
If you have replicas: 3 and volumeClaimTemplates named mysql-data, Kubernetes will create:

mysql-data-mysql-statefulset-0

mysql-data-mysql-statefulset-1

mysql-data-mysql-statefulset-2

Each pod gets its own **dedicated PVC** → which maps to a PV.

**2. What about the PVs?**

* You didn’t create a PV manually. That’s fine.
* If your cluster has a **default StorageClass**, Kubernetes will dynamically provision a PV for each PVC.
* If you don’t have a default StorageClass, the PVCs will remain in **Pending** state.

Check with:

kubectl get sc

If you see something like standard (default) → you’re good (dynamic provisioning will kick in).

**3. What will happen in your case**

* StatefulSet runs → creates 3 pods.
* Each pod requests storage via volumeClaimTemplates.
* 3 PVCs will be created (ReadWriteOnce, 500Mi each).
* If you have a default StorageClass → 3 PVs will be dynamically created and bound.
* If you don’t have one → PVCs will be stuck in Pending → Pods won’t come up.

**4. Check PVCs**

After applying your StatefulSet:

kubectl get pvc -n statefulset

You should see something like:

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

mysql-data-mysql-statefulset-0 Bound pvc-xxxx-xxx 500Mi RWO standard 1m

mysql-data-mysql-statefulset-1 Bound pvc-yyyy-yyy 500Mi RWO standard 1m

mysql-data-mysql-statefulset-2 Bound pvc-zzzz-zzz 500Mi RWO standard 1m

**5. If you don’t have a StorageClass**

* You’ll need to create either:
  + A **StorageClass** for dynamic provisioning, OR
  + A **manual PV** that matches your PVC request (size, access mode, storage class).

Example of a simple manual PV:

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-pv

spec:

capacity:

storage: 500Mi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data/mysql"

✅ **Summary:**

* You don’t need to manually create PVCs in a StatefulSet → Kubernetes does it.
* PVs will be created automatically if you have a **default StorageClass**.
* If no StorageClass exists → your PVCs will get stuck.