Lesson: Volumes, PVs, PVCs & StorageClasses

What you'll learn

- The difference between ephemeral storage that dies with a Pod and persistent storage that survives.
- What PersistentVolumes (PV), PersistentVolumeClaims (PVC), and StorageClasses are.
- Access modes RWO vs RWX and why they matter.
- Reclaim policies, and how the lab's NFS RWX StorageClass works.

Skill gained: you can give a workload storage that survives Pod restarts and rescheduling.

The lesson

By default, anything a container writes to its filesystem is **gone** when that container is replaced — and Pods get replaced all the time. For databases, uploads, and any data you must keep, you need storage that outlives the Pod. Kubernetes models this with three objects: **PersistentVolume**, **PersistentVolumeClaim**, and **StorageClass**.

1. Ephemeral vs persistent

```
Ephemeral (default + emptyDir)        Persistent (PV/PVC)
------------------------------        -------------------
lives only as long as the Pod         outlives the Pod
gone on restart / reschedule          survives restart / reschedule
scratch space, caches                 databases, uploads, anything to keep
```

You already met `emptyDir` (the shared scratch volume in the multi-container lesson). It is still **ephemeral**: when the Pod is deleted, the `emptyDir` is deleted too. For data that must persist, you need a real volume backed by external storage.

2. PV, PVC, and StorageClass

Three objects, with a clear division of labour:

- **PersistentVolume (PV)** — an actual piece of storage in the cluster (a chunk of NFS, a cloud disk). It is infrastructure.
- **PersistentVolumeClaim (PVC)** — a *request* for storage by your app: "I need 5Gi, read-write." Your Pod references the PVC, not the PV directly.
- **StorageClass** — a recipe that **provisions a PV automatically** when a PVC asks for it ("dynamic provisioning"), so an admin doesn't pre-create disks by hand.

```
You write:  PVC ("I want 5Gi RWX")
                 |
                 | references a StorageClass
                 v
   StorageClass  --dynamically creates-->  PV (real NFS storage)
                 |                            ^
                 +----------- bound -----------+
                 |
   Pod mounts the PVC -> gets the storage
```

The flow is: you create a **PVC**; the **StorageClass** provisions a matching **PV**; the PVC **binds** to it; your **Pod mounts the PVC**. You mostly only ever write the PVC and reference it.

3. A PVC and a Pod that uses it

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data
spec:
  accessModes:
    - ReadWriteMany           # see section 4
  storageClassName: nfs       # the lab's NFS StorageClass
  resources:
    requests:
      storage: 1Gi
```

```
apiVersion: v1
kind: Pod
metadata:
  name: writer
spec:
  volumes:
```

```
    - name: data
      persistentVolumeClaim:
        claimName: data        # reference the PVC by name
  containers:
    - name: writer
      image: busybox:1.36
      command: ["sh", "-c", "echo persisted >> /data/log.txt; sleep 3600"]
      volumeMounts:
        - name: data
          mountPath: /data
```

```
kubectl apply -f pvc.yaml
kubectl get pvc                 # STATUS should become Bound
kubectl get pv                  # the PV the StorageClass created
kubectl apply -f writer.yaml
kubectl exec writer -- cat /data/log.txt
# delete the Pod, recreate it -> /data/log.txt is still there
```

That last step is the whole point: delete and recreate the Pod, and the file survives because it lives on the PV, not in the Pod.

4. Access modes: RWO vs RWX

Access modes say how many nodes can mount the volume for writing at once:

- **ReadWriteOnce (RWO)** — mountable read-write by Pods on **one node**. Typical for block disks and most single-instance databases.
- **ReadWriteMany (RWX)** — mountable read-write by Pods on **many nodes** at the same time. Needed when several Pods, possibly on different workers, must share the same files. NFS supports this.
- **ReadOnlyMany (ROX)** — many nodes, read-only.

```
RWO  one node writes      [ disk ] <- Pod (node A only)
RWX  many nodes write     [ NFS  ] <- Pod(node A), Pod(node B), Pod(node C)
```

This matters: if you give a 3-replica Deployment a single RWO claim, the replicas may land on different nodes and fail to mount. For shared read-write across replicas, you need **RWX**.

5. The lab's NFS RWX StorageClass

The lab provides persistent storage from an **NFS server at 10.100.100.12**, exposed as a StorageClass that supports **ReadWriteMany (RWX)**. So when you write a PVC with `storageClassName` set to the lab's NFS class and `accessModes: [ReadWriteMany]`, a PV is provisioned on that NFS server and your Pods — even across the three workers (.8/.9/.10) — can all read and write the same data.

Check what StorageClass name to use:

```
kubectl get storageclass            # note the NFS class name (use it in your PVC)
kubectl describe storageclass <name>
```

Use that exact name in `storageClassName`. RWX from NFS is what lets multiple Pods share files in this lab — keep that in mind for the assignment.

6. Reclaim policy — what happens when the PVC is deleted

A PV has a **reclaim policy** that decides the fate of the data when its PVC is deleted:

- **Delete** — the PV and its underlying storage are deleted too. Data gone. Convenient, dangerous.
- **Retain** — the PV (and data) is kept; an admin must clean it up manually. Safer for important data.

```
kubectl get pv -o custom-
columns=NAME:.metadata.name,POLICY:.spec.persistentVolumeReclaimPolicy
```

For anything you care about, prefer **Retain** so an accidental `kubectl delete pvc` doesn't wipe your data.

7. StatefulSets and storage

Recall the StatefulSet from the Workloads lesson: its `volumeClaimTemplates` automatically create **one PVC per Pod** (`data-db-0`, `data-db-1`, …), each binding its own PV. That is how each database replica keeps its own private, persistent disk. The lab NFS StorageClass can back those claims too.

```
kubectl get pvc                 # for a StatefulSet you'll see one PVC per Pod ordinal
```

The takeaway: write a **PVC**, point it at the right **StorageClass** (NFS for RWX in this lab), choose the right **access mode**, and your data survives Pod churn. You will prove this end to end in Assignment 2.

Dig deeper

- Persistent Volumes
- Storage Classes
- Volumes
- Dynamic Volume Provisioning

Search terms

- `kubernetes ephemeral vs persistent volume`
- `kubernetes pv pvc storageclass explained`
- `kubernetes access modes ReadWriteOnce ReadWriteMany`
- `kubernetes nfs storageclass dynamic provisioning`
- `kubernetes persistent volume reclaim policy retain delete`

Check yourself

1. What happens to data written to a container's normal filesystem when the Pod is replaced?
2. Describe the roles of PV, PVC, and StorageClass and how they connect.
3. What is the difference between RWO and RWX, and which does the lab NFS class provide?
4. Why might a 3-replica Deployment with a single RWO claim fail?
5. What does the `Retain` reclaim policy do when you delete a PVC, and why prefer it for important data?

Revision #4
Created 2026-05-24 16:49:00 UTC by Eslam Shapsough
Updated 2026-05-24 16:53:00 UTC by Eslam Shapsough