Chapter 11

Kubernetes Volume Management

Introduction

- To back a Pod with a persistent storage, Kubernetes uses **Volumes.** In this chapter, we will learn about Volumes and their types. We will also talk
  about **PersistentVolume** and **PersistentVolumeClaim** objects, which help us attach storage Volumes to Pods.

Learning Objectives

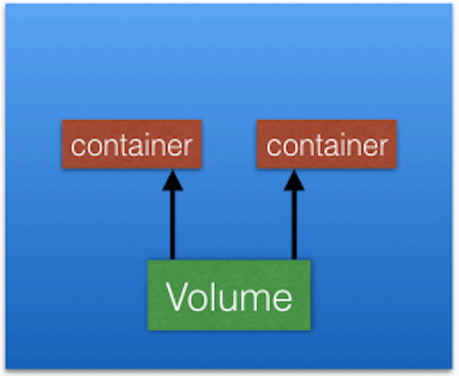
By the end of this chapter, you should be able to:

- Explain the need for persistent data management.
- Discuss Kubernetes Volume and its types.
- Discuss PersistentVolumes and PersistentVolumeClaims.

Volumes

As we know, containers, which create the Pods, are ephemeral in nature. **All data stored inside a container is deleted if the container crashes**. However, the kubelet will restart it with a clean state, which means that it will **not have any of the old data.**

To overcome this problem, Kubernetes uses Volumes. A Volume is essentially a directory backed by a storage medium. The storage medium and its content are determined by the Volume Type.



Volumes

In Kubernetes, **a Volume is attached to a Pod and shared among the containers of that Pod**. The Volume has the same life span as the Pod, and it outlives the containers of the Pod - this allows data to be preserved across container restarts.

Volume Types

A directory which is mounted inside a Pod is backed by the underlying Volume Type. A Volume Type decides the properties of the directory, like size, content, etc. Some examples of Volume Types are:

- **`emptyDir`** An **`empty`** Volume is created for the Pod as soon as it is scheduled on the worker node. The Volume's life is tightly coupled with the Pod. If the Pod dies, the content of **`emptyDir`** is deleted forever.

- **`hostPath`** With the **`hostPath`** Volume Type, we can share a directory from the host to the Pod. If the Pod dies, the content of the Volume is still available on the host.

- **`gcePersistentDisk`** With the **`gcePersistentDisk`** Volume Type, we can mount a Google Compute Engine (GCE) persistent disk into a Pod.

- **`awsElasticBlockStore`** With the **`awsElasticBlockStore`** Volume Type, we can mount an AWS EBS Volume into a Pod.

- **`Nfs`** With nfs, we can mount an NFS share into a Pod.

- **`Iscsi`** With iscsi, we can mount an iSCSI share into a Pod.

- **`Secret`** With the **`secret`** Volume Type, we can pass sensitive information, such as passwords, to Pods.

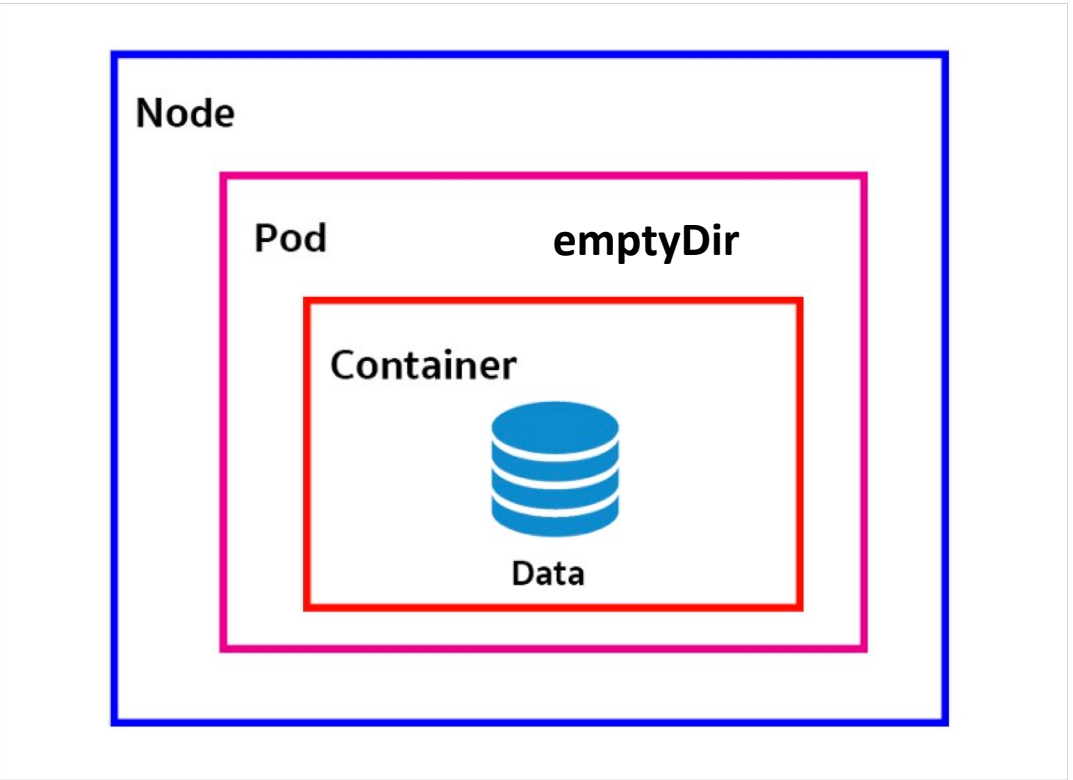
Volume Types

Temp	Local	Network
emptyDir	hostPath	GlusterFS gitRepo NFS iSCSI gcePersistentDisk AWS EBS azureDisk Fiber Channel Secret VshereVolume

Persistent Volume Phase

- Available - a free resource that yet is not bound to a claim
- Bound - the volume is bound to a claim
- Released - the claim has been deleted but the resource is not yet reclaimed by the cluster
- Failed - the volume has failed its automatic reclamation

emptyDir

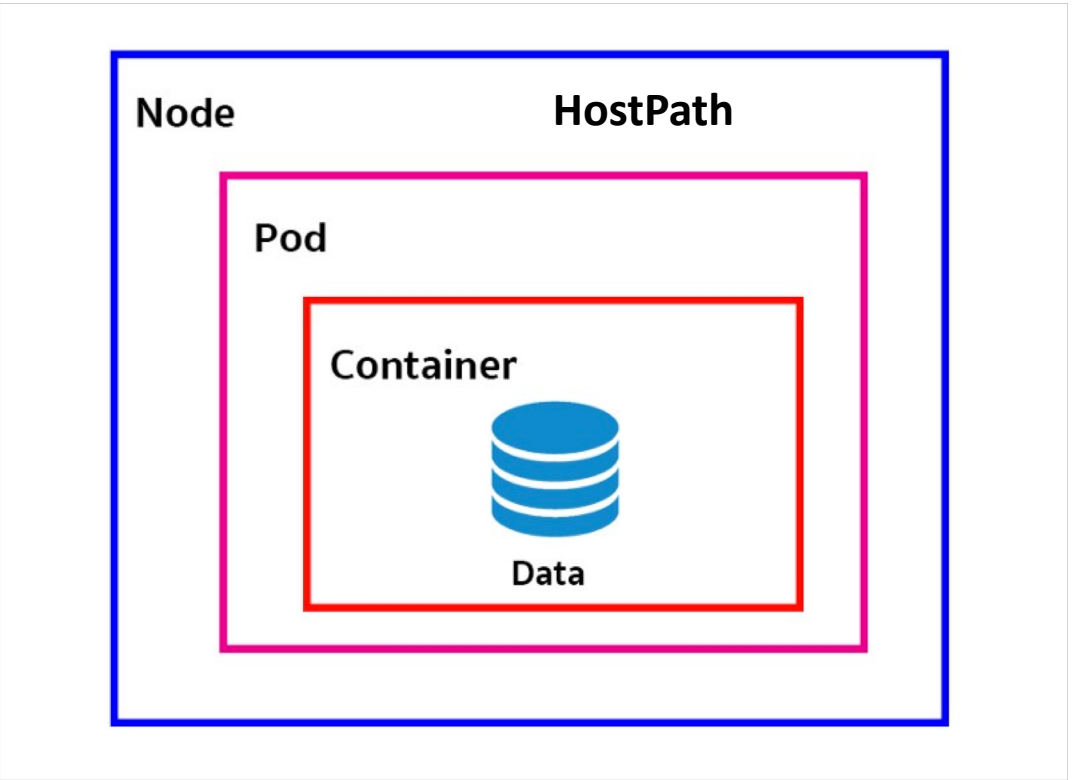



```
apiVersion: v1
kind: Pod
metadata:
name: test-pd
spec:
containers:
- image: k8s.gcr.io/test-webserver
name: test-container
volumeMounts:
- mountPath: /cache
name: cache-volume
volumes:
- name: cache-volume
emptyDir: {}
```

ข้อควรระวัง

- เมื่อ Pod ถูกลบจาก Node ข้อมูลใน emptyDir จะถูกลบออกไปอย่างถาวร

Hostpath



ข้อควรระวัง

ต้องมั่นใจว่า Pod ที่เราต้องการจะรัน ทำงานอยู่บน Node ที่เราเตรียมข้อมูลไว้แล้ว มิฉะนั้น Pod จะหา Volume ไม่เจอ

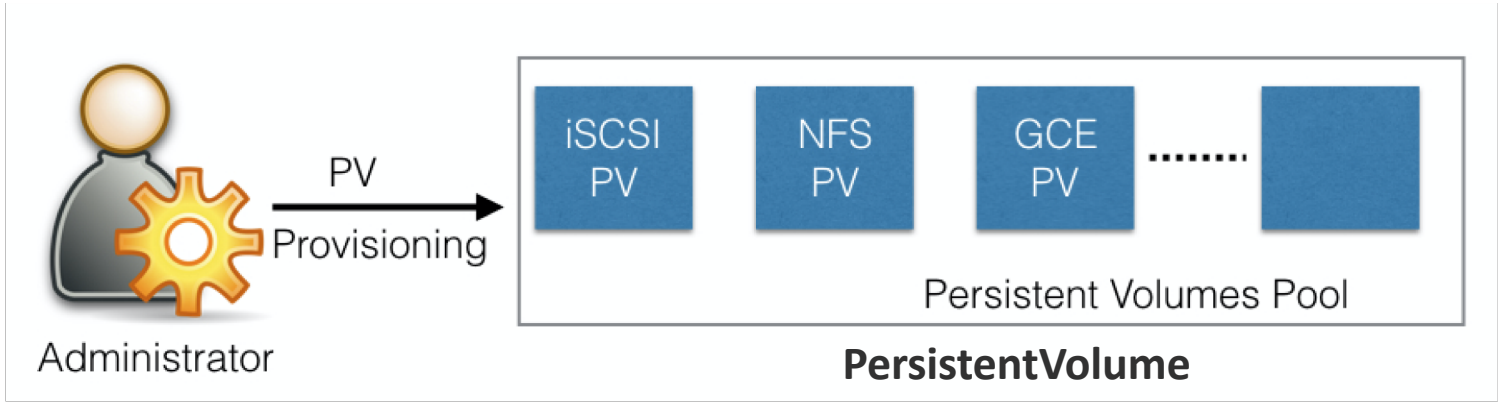
```
apiVersion: v1
kind: Pod
metadata:
name: test-pd
spec:
containers:
- image: k8s.gcr.io/test-webserver
name: test-container
volumeMounts:
- mountPath: /test-pd
name: test-volume
volumes:
- name: test-volume
hostPath:
# directory location on host
path: /data
# this field is optional
type: Directory
```

PersistentVolumes

In a typical IT environment, storage is managed by the storage/system administrators. The end user will just get instructions to use the storage, but does not have to worry about the underlying storage management.

In the containerized world, we would like to follow similar rules, but it becomes challenging, given the many Volume Types we have seen earlier. Kubernetes resolves this problem with the **PersistentVolume (PV)** subsystem, which provides APIs for users and administrators to manage and consume storage. **To manage the Volume, it uses the PersistentVolume API resource type**, and to **consume it, it uses the PersistentVolumeClaim API resource type**.

A Persistent Volume is a network-attached storage in the cluster, which is provisioned by the administrator.



PersistentVolumes

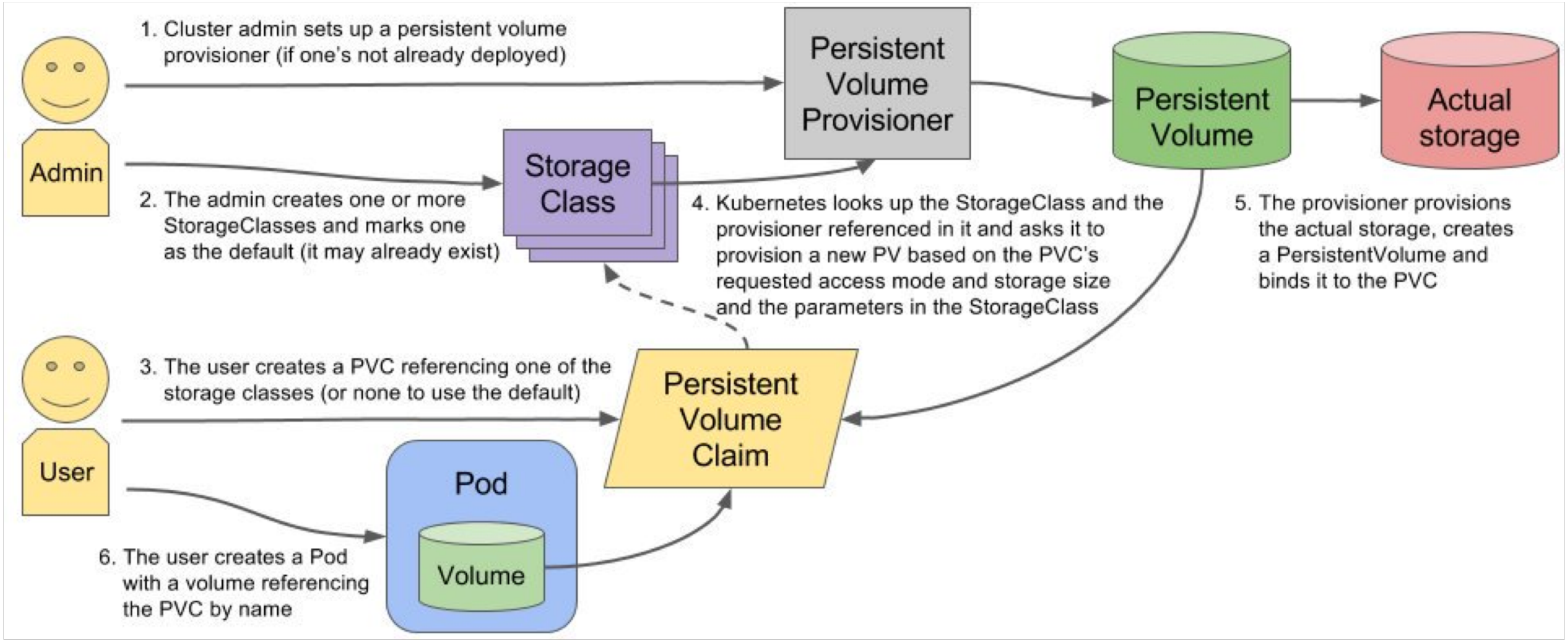
PersistentVolumes can be dynamically provisioned based on the StorageClass resource. A StorageClass contains pre-defined provisioners and parameters to create a PersistentVolume. Using PersistentVolumeClaims, a user sends the request for dynamic PV creation, which gets wired to the StorageClass resource.

Some of the Volume Types that support managing storage using PersistentVolumes are:

- GCEPersistentDisk
- AWSElasticBlockStore
- AzureFile
- NFS
- iSCSI.

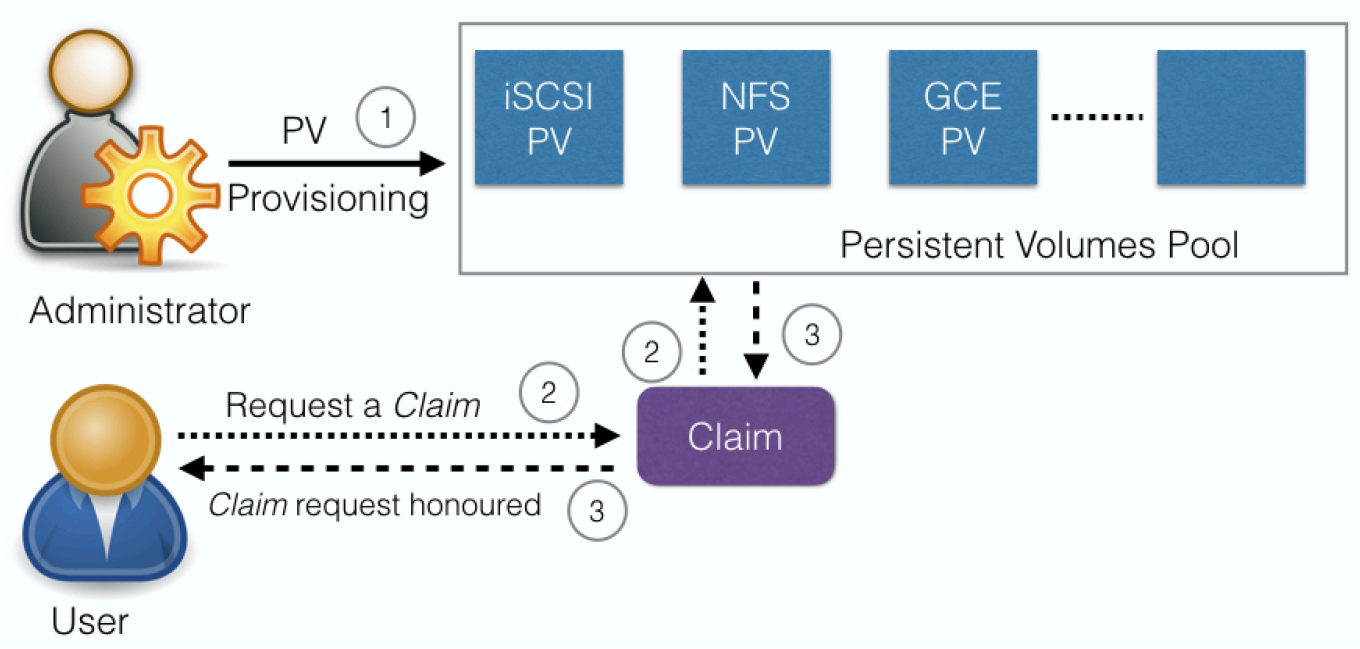
For a complete list, as well as more details, you can check out the Kubernetes documentation.

Dynamic Persistent Volume Provisioning



PersistentVolumeClaims

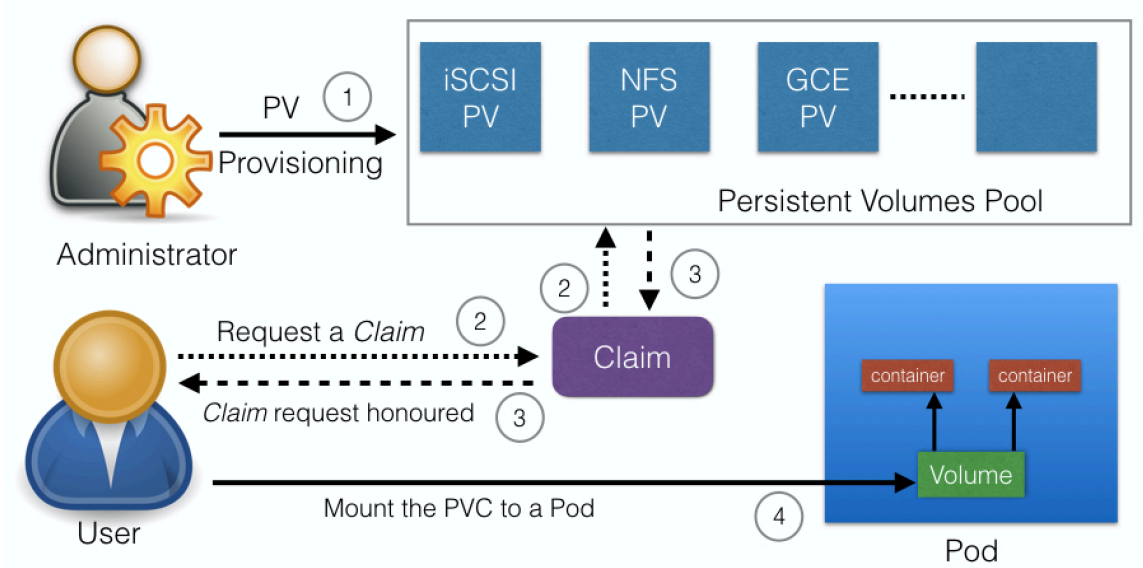
A **PersistentVolumeClaim (PVC)** is a request for storage by a user. Users request for PersistentVolume resources based on size, access modes, etc. Once a suitable PersistentVolume is found, it is bound to a PersistentVolumeClaim.



PersistentVolumeClaim

PersistentVolumeClaims

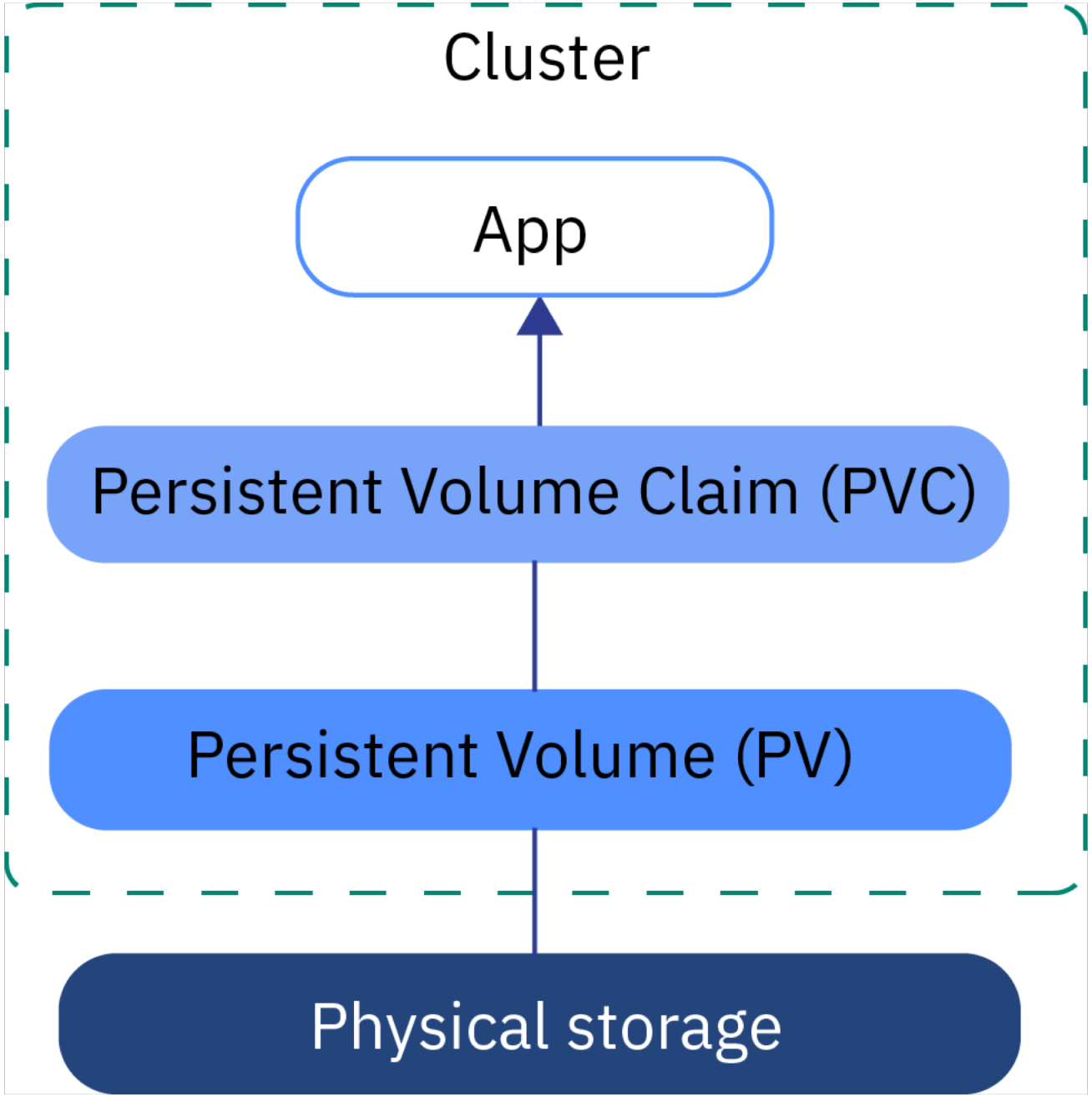
After a successful **bound**, the PersistentVolumeClaim resource can be used in a Pod.



PersistentVolumeClaim Used In a Pod

Once a user finishes its work, the attached PersistentVolumes can be released. The underlying PersistentVolumes can then be **reclaimed** and **recycled** for future usage.

To learn more, you can check out the Kubernetes documentation.


Cluster
App
Persistent Volume Claim (PVC)
Persistent Volume (PV)
Physical storage

Access Mode

This table outlines the different access modes currently available in Kubernetes for PVs:

Name	Abbr.	Use Case
ReadWriteOnce	RWO	Most common model. High performance stateful workload without need for shared filesystem access between compute nodes. The PV can be mounted read-write on one compute node at a time.
ReadWriteMany	RWX	Typically served to a container over NFS when a workload require both read and write access to shared unstructured data. The PV can be mounted read-write by multiple compute nodes at a time.
ReadOnlyMany	ROX	In most cases read-only access to NFS exports. The PV can be mounted read-only by multiple compute nodes.

Recycling

- When a user is done with their volume, they can delete PVC object which allows reclamation of the resource
- Reclaim policy for PersistentVolume tell the cluster what to do with the volume after it has been released of its claim
- Reclaim policy
  - Retain (remain / cannot reuse)
    After PVC is deleted, PV still exists and volume is marked as “released”. It cannot be reused by other PVC. (it needs to delete manually)
  - Delete
    Delete volume as well as associated storage asset in external infrastructure
  - Recycle (remain / can reuse)
    Perform basic scrub (rm -f /thevolume/*) on 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.

Lab

- Lab 4 : Challenge !!

Learning Objectives (Review)

You should now be able to:

- Explain the need for persistent data management.
- Discuss Kubernetes Volume and its types.
- Discuss PersistentVolumes and PersistentVolumeClaims.

Refererence

https://www.slideshare.net/Byungwook/kubernetes-4-volume-amp-stateful-set