# Kubernetes - Volumes

In Kubernetes, a volume can be thought of as a directory which is accessible to the containers in a pod. We have different types of volumes in Kubernetes and the type defines how the volume is created and its content.

The concept of volume was present with the Docker, however the only issue was that the volume was very much limited to a particular pod. As soon as the life of a pod ended, the volume was also lost.

On the other hand, the volumes that are created through Kubernetes is not limited to any container. It supports any or all the containers deployed inside the pod of Kubernetes. A key advantage of Kubernetes volume is, it supports different kind of storage wherein the pod can use multiple of them at the same time.

### Types of Kubernetes Volume

Here is a list of some popular Kubernetes Volumes −

**emptyDir** − It is a type of volume that is created when a Pod is first assigned to a Node. It remains active as long as the Pod is running on that node. The volume is initially empty and the containers in the pod can read and write the files in the emptyDir volume. Once the Pod is removed from the node, the data in the emptyDir is erased.

**hostPath** − This type of volume mounts a file or directory from the host node’s filesystem into your pod.

**gcePersistentDisk** − This type of volume mounts a Google Compute Engine (GCE) Persistent Disk into your Pod. The data in a **gcePersistentDisk** remains intact when the Pod is removed from the node.

**awsElasticBlockStore** − This type of volume mounts an Amazon Web Services (AWS) Elastic Block Store into your Pod. Just like **gcePersistentDisk**, the data in an **awsElasticBlockStore** remains intact when the Pod is removed from the node.

**nfs** − An **nfs** volume allows an existing NFS (Network File System) to be mounted into your pod. The data in an **nfs** volume is not erased when the Pod is removed from the node. The volume is only unmounted.

**iscsi** − An **iscsi** volume allows an existing iSCSI (SCSI over IP) volume to be mounted into your pod.

**flocker** − It is an open-source clustered container data volume manager. It is used for managing data volumes. A **flocker** volume allows a Flocker dataset to be mounted into a pod. If the dataset does not exist in Flocker, then you first need to create it by using the Flocker API.

**glusterfs** − Glusterfs is an open-source networked filesystem. A glusterfs volume allows a glusterfs volume to be mounted into your pod.

**rbd** − RBD stands for Rados Block Device. An **rbd** volume allows a Rados Block Device volume to be mounted into your pod. Data remains preserved after the Pod is removed from the node.

**cephfs** − A **cephfs** volume allows an existing CephFS volume to be mounted into your pod. Data remains intact after the Pod is removed from the node.

**gitRepo** − A **gitRepo** volume mounts an empty directory and clones a **git** repository into it for your pod to use.

**secret** − A **secret** volume is used to pass sensitive information, such as passwords, to pods.

**persistentVolumeClaim** − A **persistentVolumeClaim** volume is used to mount a PersistentVolume into a pod. PersistentVolumes are a way for users to “claim” durable storage (such as a GCE PersistentDisk or an iSCSI volume) without knowing the details of the particular cloud environment.

**downwardAPI** − A **downwardAPI** volume is used to make downward API data available to applications. It mounts a directory and writes the requested data in plain text files.

**azureDiskVolume** − An **AzureDiskVolume** is used to mount a Microsoft Azure Data Disk into a Pod.

## Persistent Volume and Persistent Volume Claim

**Persistent Volume (PV)** − It’s a piece of network storage that has been provisioned by the administrator. It’s a resource in the cluster which is independent of any individual pod that uses the PV.

**Persistent Volume Claim (PVC)** − The storage requested by Kubernetes for its pods is known as PVC. The user does not need to know the underlying provisioning. The claims must be created in the same namespace where the pod is created.

### Creating Persistent Volume

kind: PersistentVolume ---------> 1

apiVersion: v1

metadata:

name: pv0001 ------------------> 2

labels:

type: local

spec:

capacity: -----------------------> 3

storage: 10Gi ----------------------> 4

accessModes:

- ReadWriteOnce -------------------> 5

hostPath:

path: "/tmp/data01" --------------------------> 6

In the above code, we have defined −

**kind: PersistentVolume** → We have defined the kind as PersistentVolume which tells kubernetes that the yaml file being used is to create the Persistent Volume.

**name: pv0001** → Name of PersistentVolume that we are creating.

**capacity:** → This spec will define the capacity of PV that we are trying to create.

**storage: 10Gi** → This tells the underlying infrastructure that we are trying to claim 10Gi space on the defined path.

**ReadWriteOnce** → This tells the access rights of the volume that we are creating.

**path: "/tmp/data01"** → This definition tells the machine that we are trying to create volume under this path on the underlying infrastructure.

### Creating PV

$ kubectl create –f local-01.yaml

persistentvolume "pv0001" created

### Checking PV

$ kubectl get pv

NAME CAPACITY ACCESSMODES STATUS CLAIM REASON AGE

pv0001 10Gi RWO Available 14s

### Describing PV

$ kubectl describe pv pv0001

### Creating Persistent Volume Claim

kind: PersistentVolumeClaim --------------> 1

apiVersion: v1

metadata:

name: myclaim-1 --------------------> 2

spec:

accessModes:

- ReadWriteOnce ------------------------> 3

resources:

requests:

storage: 3Gi ---------------------> 4

In the above code, we have defined −

**kind: PersistentVolumeClaim** → It instructs the underlying infrastructure that we are trying to claim a specified amount of space.

**name: myclaim-1** → Name of the claim that we are trying to create.

**ReadWriteOnce** → This specifies the mode of the claim that we are trying to create.

**storage: 3Gi** → This will tell kubernetes about the amount of space we are trying to claim.

### Creating PVC

$ kubectl create –f myclaim-1

persistentvolumeclaim "myclaim-1" created

### Getting Details About PVC

$ kubectl get pvc

NAME STATUS VOLUME CAPACITY ACCESSMODES AGE

myclaim-1 Bound pv0001 10Gi RWO 7s

### Describe PVC

$ kubectl describe pv pv0001

### Using PV and PVC with POD

kind: Pod

apiVersion: v1

metadata:

name: mypod

labels:

name: frontendhttp

spec:

containers:

- name: myfrontend

image: nginx

ports:

- containerPort: 80

name: "http-server"

volumeMounts: ----------------------------> 1

- mountPath: "/usr/share/tomcat/html"

name: mypd

volumes: -----------------------> 2

- name: mypd

persistentVolumeClaim: ------------------------->3

claimName: myclaim-1

In the above code, we have defined

**volumeMounts:** → This is the path in the container on which the mounting will take place.

**Volume:** → This definition defines the volume definition that we are going to claim.

**persistentVolumeClaim:** → Under this, we define the volume name which we are going to use in the defined pod.