
kubernetes
STORAGE

❖ KUBERNETES-VOLUMES:

- Volumes in Kubernetes is a directory which is accessible to the containers in a pod.
- Volumes in Kubernetes allow user to Store Data Outside the Container.
- 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 VOLUMES:

- awsElasticBlockStore
- azureFile
- cephfs
- emptyDir
- fc (fibre channel)
- gitRepo
- iscsi
- nfs
- gcePersistentDisk
- azureDisk
- cinder
- hostPath
- flocker
- glusterfs
- local
- vsphereVolume....etc

→ **To list volume types:**

$kubectl explain pod.spec.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.

EXAMPLE: A SINGLE CONTAINER CONFIGURATION:

```
apiVersion: v1
kind: Pod
metadata:
  name: emptydir-one-container
spec:
  containers:
  - image: centos:7
    command:
      - sleep
      - "3600"
    name: test-container
    volumeMounts:
    - mountPath: /tmp
      name: tmp-volume
  volumes:
  - name: tmp-volume
    emptyDir: {}
```

```
$kubectl create -f emptydir-pod.yaml
$kubectl get po
$kubectl exec -it emptydir-one-container -c test-container /bin/sh
$kubectl exec -it test-pod --ls -l /tmp/file1
```

EXAMPLE: EMPTYDIR IN MULTIPLE CONTAINERS:

```
$vim emptydir-pod-two.yaml
apiVersion: v1
kind: Pod
metadata:
  name: emptydir-two-containers
spec:
  containers:
  - name: centos1
    image: centos:7
    command:
      - sleep
      - "3600"
    volumeMounts:
      - mountPath: /centos1
        name: tmp-volume
  - name: centos2
    image: centos:7
    command:
      - sleep
      - "3600"
    volumeMounts:
      - mountPath: /centos2
        name: tmp-volume
 volumes:
    - name: tmp-volume
      emptyDir: {}
```

$kubectl get po

$kubectl exec -it emptydir-two-containers -c centos1 /bin/bash

$touch /centos1/aws

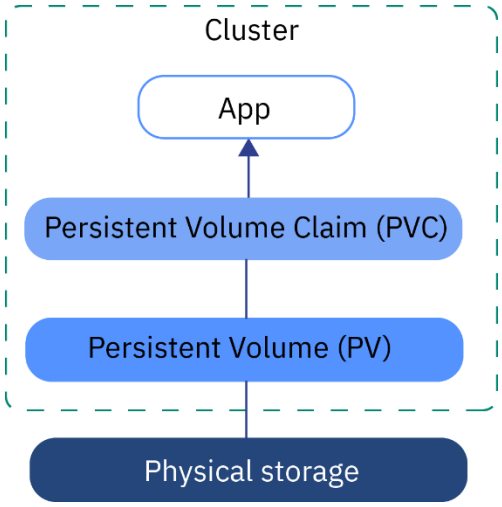
$ls /centos1

$kubectl exec -it emptydir-two-containers -c centos2 /bin/bash

$ls /centos2 [we will see aws file here]

➢ HOSTPATH:

- This type of volume mounts a file or directory from the host node's filesystem into your pod.



PHYSICAL STORAGE:

- A physical storage instance that you can use to persist your data.

PERSISTENT VOLUMES (PV):

- A Persistent Volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.
- It is a resource in the cluster just like a node is a cluster resource.

PERSISTENTVOLUMECLAIM (PVC):

- A PersistentVolumeClaim (PVC) is a request for storage by a user.
- It is similar to a Pod. Pods consume node resources and PVCs consume PV resources.
- Pods can request specific levels of resources (CPU and Memory).
- Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany.

CLUSTER:

- By default, every cluster is set up with a plug-in to provision file storage.
- You can choose to install other add-ons, such as the one for block storage.
- To use storage in a cluster, you must create a persistent volume claim, a persistent volume and a physical storage instance.

APP:

- To read from and write to your storage instance, you must mount the persistent volume claim (PVC) to your app.

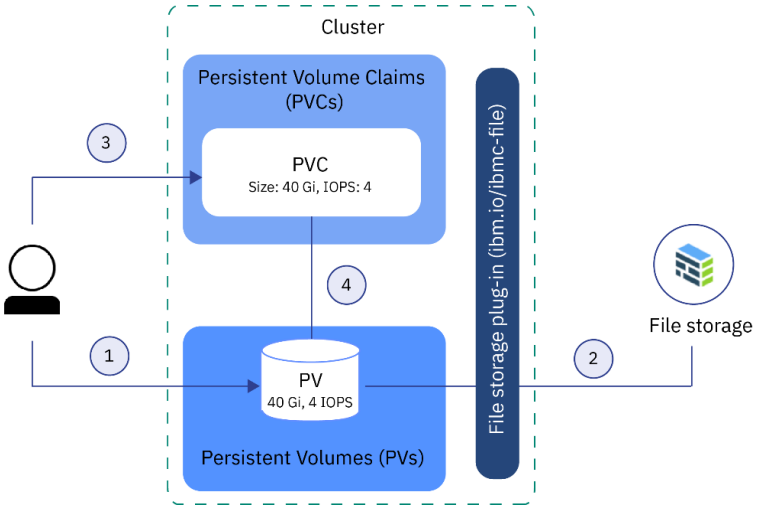
  NOTE: Different storage types have different read-write rules.

  For example, you can mount multiple pods to the same PVC for file storage. Block storage comes with a RWO (ReadWriteOnce) access mode so that you can mount the storage to one pod only.

➢ **PROVISIONING TYPES:** Kubernetes supports static and dynamic.

STATIC PROVISIONING:

- In the static provisioning, PVs are created by the cluster administrator and are allowed to use actual storage available in the cluster. This means that in order to use static provisioning, one needs to have a storage (e.g., Amazon EBS) capacity provisioned beforehand.

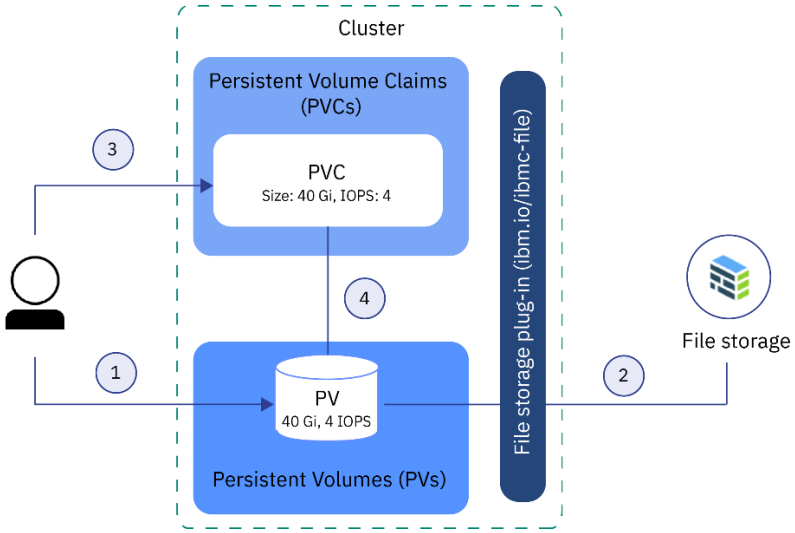


Sample flow for static provisioning of file storage

1. The cluster admin gathers all the details about the existing storage device and creates a persistent volume (PV) in the cluster.
2. 2. Based on the storage details in the PV, the storage plug-in connects the PV with the storage device in your IBM Cloud infrastructure account.
3. The cluster admin or a developer creates a PVC. Because the PV and the storage device already exist, no storage class is specified in the PVC.
4. After the PVC is created, the storage plug-in tries to match the PVC to an existing PV. The PVC and the PV match when the same values for the size, IOPS, and access mode are used in the PVC and the PV. When PVC and PV match, the status of the PVC and the PV changes to Bound. You can now use the PVC to mount persistent storage to your app. When you delete the PVC, the PV and the physical storage instance are not removed. You must remove the PVC, PV, and the physical storage instance separately.

DYNAMIC PROVISIONING:

- A dynamic provisioning of volumes can be triggered when a volume type claimed by the user does not match any PVs available in the cluster. Sample flow for dynamic provisioning of file storage with the pre-defined silver storage class.



Sample flow for static provisioning of file storage:

1. The user creates a persistent volume claim (PVC) that specifies the storage type, storage class, size in gigabytes, number of IOPS, and billing type. The storage class determines the type of storage that is provisioned and the allowed ranges for size and IOPS. Creating a PVC in a cluster automatically triggers the storage plug-in for the requested type of storage to provision storage with the given specification.
2. The storage device is automatically ordered and provisioned into your IBM Cloud infrastructure account. The billing cycle for your storage device starts.
3. The storage plug-in automatically creates a persistent volume (PV) in the cluster, a virtual storage device that points to the actual storage device in your IBM Cloud infrastructure account.
4. The PVC and PV are automatically connected to each other. The status of the PVC and the PV changes to Bound. You can now use the PVC to mount persistent storage to your app. If you delete the PVC, the PV and related storage instance are also deleted.

CREATING A PV:

$vim pv.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  storageClassName: local-storage
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

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

CREATING A PVC:

$vim pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

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

CREATING A POD:

$vim pvc-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: pvc-pod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["/bin/sh", "-c", "while true; do sleep 3600; done"]
    volumeMounts:
    - mountPath: "/mnt/storage"
      name: my-storage
  volumes:
  - name: my-storage
    persistentVolumeClaim:
      claimName: my-pvc
```

```
$kubectl create -f pvc-pod.yaml
$kubectl get po
$kubectl exec -it pvc-pod -- /bin/sh
$cd /mnt/storage
$touch abc
under hostpath (node1 or node2)
$cd /mnt/data
$ls
```