Volumes

Why do we need Volumes?

Storage volumes or persistent volumes, are needed because there's no data saved within Kubernetes. If you delete a pod, the data is lost. You must setup storage that doesn't depend on the pod life-cycle. This way, new pods can pick up where the old one left off and storage will be available to the node for access. Volumes are managed at the node level.

▶ Persistent Volume (PV)

If you think of a U haul Storage facility the PV is the storage locker – this is where the data actually resides. This is the storage that defines the size, usage, readwrite permissions, and reclaimPolicy.

▶ Persistent Volume Claim(PVC)

A PVC is the form you fill out at the U-Haul store that requests to rent a storage unit. Its defines the resources needed for a pod to stake claim to storage, memory, and CPU of volume. Understand that this is a **claim, not a guarantee.**

Reclaim Mode – Defines what to do after a user is done with their volume.

Retain – policy allows for manual reclamation of the resource. The storage is considered 'released', but not available for another claim.

Deleted – removes the PV from Kubernetes and the external storage asset -AWS EBS, GCE, Azure Disk,etc

Access Mode – The PVC will also define access modes which define permissions for your volume:

ReadWriteOnce – Only a single pod can read and write data.
ReadOnlyMany – Many nodes can read data.
ReadWriteMany – Many nodes can read and write data.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

▶ Storage Classes

A PVC is either administered or dynamically provisioned using storage classes. Storage classes are options that are available to a business to store their data. For more traditional businesses, they may want to connect to their existing storage area network (SAN) using iSCSI as their connection to the cloud. Or they may use a network file system (NFS) for distributed storage to employees within a company.

Most modern cloud deployments look something like this when connecting to their cloud provider:

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp3
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

https://kubernetes.io/docs/concepts/storage/storage-classes/

▶ Local Volumes

Local volumes are a quick solution for saving data that simply creates a volume on a node within your cluster. Its a convenient method, however they are not tied to one specific node and they do not survive cluster crashes. **Configmaps** and secrets are considered local volumes within Kubernetes.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

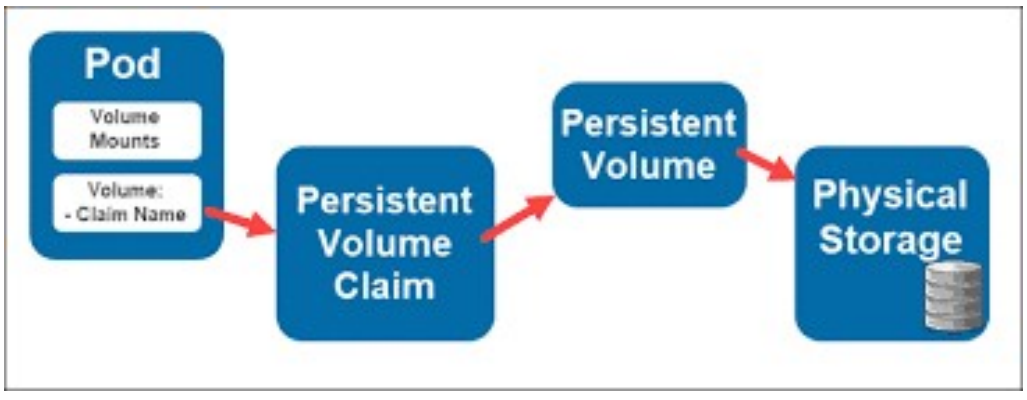
Summary

▶ Who creates the Persistent Volumes?

As Kubernetes admins, your job is to setup and maintain clusters and also to ensure users have the resources they need. If customers or users, deploy applications into a cluster you've created for them, it's your job to manage storage requirements. Therefore, storage is managed by administrators.

▶ Order of Operations to Select a Volume

1. First the pod requests a volume with the the volume claim.
2. The claim then tries to find a volume in the cluster, that meets the claims requests.
3. Once a volume is found it then accesses the actual storage back end (Cloud, NFS, etc).



Hands On – Configure a Pod to use a Persistent Volume

I. Setup your node

We first need to create a directory on our node that will point us to our volume. In order to this we need to ssh into our node, which in this case is minikube.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get node
NAME       STATUS   ROLES                  AGE     VERSION
minikube   Ready    control-plane,master   3m44s   v1.22.3
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ minikube ssh
Last login: Wed Mar 16 23:54:29 2022 from 192.168.49.1
docker@minikube:~$
```

Create the directory and an index.html within it as an anchor. After this successful, **exit your shell.**

https://explainshell.com/

```
docker@minikube:~$ sudo mkdir /mnt/data
docker@minikube:~$ sudo sh -c "echo 'Hello from Kubernetes storage' > /mnt/data/index.html"
docker@minikube:~$ cat /mnt/data/index.html
Hello from Kubernetes storage
docker@minikube:~$
```

```
docker@minikube:~$ exit
logout
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

II. Create a Persistent Volume (local)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <yourname-storage>
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 20Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

This configuration file specifies that the volume is at *mnt*/data and that its looking for 20MB with and access mode of ReadWriteOnce.

The storageClasssName is set to manual which means the PV and PVC will be bound together.

Begin by creating the file above and then apply the deployment to the cluster.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ nano doms-pv.yaml
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-pv.yaml
 apiVersion: v1
    kind: PersistentVolume
    metadata:
      name: doms-storage
      labels:
        type: local
    spec:
      storageClassName: manual
      capacity:
        storage: 20Mi
      accessModes:
        - ReadWriteOnce
      hostPath:
        path: "/mnt/data"
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-pv.yaml
persistentvolume/doms-storage created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

View your new PV.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pv doms-storage
NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
doms-storage   20Mi       RWO            Retain           Available           manual                  84s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

III. Create a Persistent Volume Claim

Pods use the PVC to request physical storage.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: yourname-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Mi
```

This claim is grants ReadWriteOnce access and requests 3MBs of storage space.

Create the claim, verify its configuration, and then apply it to the cluster. Once finished verify that it exists.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ nano doms-pv-claim.yaml
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-pv-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: doms-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Mi
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-pv-claim.yaml
persistentvolumeclaim/doms-pv-claim created
```

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pvc doms-pv-claim
NAME           STATUS   VOLUME         CAPACITY   ACCESS MODES   STORAGECLASS   AGE
doms-pv-claim  Bound    doms-storage   20Mi       RWO            manual         5m22s
```

IV. Create a Pod and Attach the Claim

This is the last step and often most difficult. Not only are we creating an Nginx pod, but we're also attaching the PV, PVC, and the file path.

```
apiVersion: v1
kind: Pod
metadata:
  name: yourname-pv-pod
spec:
  volumes:
    - name: yourname-storage
      persistentVolumeClaim:
        claimName: yourname-pv-claim
  containers:
    - name: yourname-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: yourname-storage
```

This pod create an Nginx server with port 80 open. It also shows the storage volume we will need, along with the volume claim we will be requesting.

Create the pod, verify its configuration, apply it to the cluster, and then make sure its working.

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ cat doms-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: doms-pv-pod
spec:
  volumes:
    - name: doms-storage
      persistentVolumeClaim:
        claimName: doms-pv-claim
  containers:
    - name: doms-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: doms-storage
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl apply -f doms-pod.yaml
pod/doms-pv-pod created
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl get pod doms-pv-pod
NAME          READY   STATUS    RESTARTS   AGE
doms-pv-pod   1/1     Running   0          54s
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```

To verify that our storage is setup correctly, we need to SSH into the new Pod with the following command:

kubectl exec -it doms-pv-pod -- /bin/bash

```
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl exec -it doms-pv-pod -- /bin/bash
root@doms-pv-pod:/# apt update
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian bullseye-updates InRelease [39.4 kB]
Get:3 http://security.debian.org/debian-security bullseye-security InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main amd64 Packages [8183 kB]
Get:5 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [2596 B]
Get:6 http://security.debian.org/debian-security bullseye-security/main amd64 Packages [121 kB]
Fetched 8506 kB in 2s (5163 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
3 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@doms-pv-pod:/# apt install curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.74.0-1.3+deb11u1).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
root@doms-pv-pod:/# curl http://localhost/
Hello from Kubernetes storage
root@doms-pv-pod:/#
```

Once we're in, we need to do a quick update using the command apt update.

We then need to install a tool call 'curl' using the apt install curl command. Curl is a basic bash tool that allows us to test the transfer of data between two ends. In this case we are using curl http://localhost to test our connection from the pod we've created to the storage directory that we originally created on the minikube node.

If you receive the message:

'Hello from Kubernetes storage'

You have successfully completed this lesson!

V. Cleanup

```
root@doms-pv-pod:/# exit
exit
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl delete pod doms-pv-pod
pod "doms-pv-pod" deleted
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl delete pvc doms-pv-claim
persistentvolumeclaim "doms-pv-claim" deleted
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$ kubectl delete pv doms-storage
persistentvolume "doms-storage" deleted
dominickhrndz314@cloudshell:~ (sandbox-io-289003)$
```