K8S (7) Volume Management

Volume is linked to Pod, has the same life span with pod. Volume is shares among the container if this pod.
Volume is a mount point in container's file system.
So when the container crush, the kubelet restart the container. The data will still be in the Volume.
(Volume lives inside the pod, shared among the container in this pod. When pod is deleted, the volume deleted, but nor affected by the container deletion)

Volume type:

Decides the property of the mounted directory:
(1) **emptyDir:** Created for the Pod as on the worker node. The Volume's life is tightly coupled with the Pod. If the Pod is terminated, the content of **emptyDir** is deleted forever.
(2) **hostPath:** The **hostPath** shares a directory between the host and the Pod. If the Pod is terminated, the content of the Volume is still available on the host.
(3) **gcePersistentDisk:** Mount a Google Compute Engine (GCE) persistent disk into a Pod.
(4) **awsElasticBlockStore**: **M**ount an AWS EBS Volume into a Pod.
(5) **azureDisk**: Mount a Microsoft Azure Data Disk into a Pod.
(6) **azureFile**: Mount a Microsoft Azure File Volume into a Pod.
(7) **cephfs**: **Mount a**n existing CephFS volume into a Pod. When a Pod terminates, the volume is unmounted and the contents of the volume are preserved.
(8) **nfs**: With **nfs**, we can mount an NFS share into a Pod.
(9) **iscsi**: With **iscsi**, we can mount an iSCSI share into a Pod.
(10) **secret**: With the **secret** Volume Type, we can pass sensitive information, such as passwords, to Pods.
(11) **configMap**: With **configMap** objects, we can provide configuration data, or shell commands and arguments into a Pod.
(12) **persistentVolumeClaim**: We can attach a **PersistentVolume** to a Pod using a **persistentVolumeClaim**.

PersistentVolumes (PV):

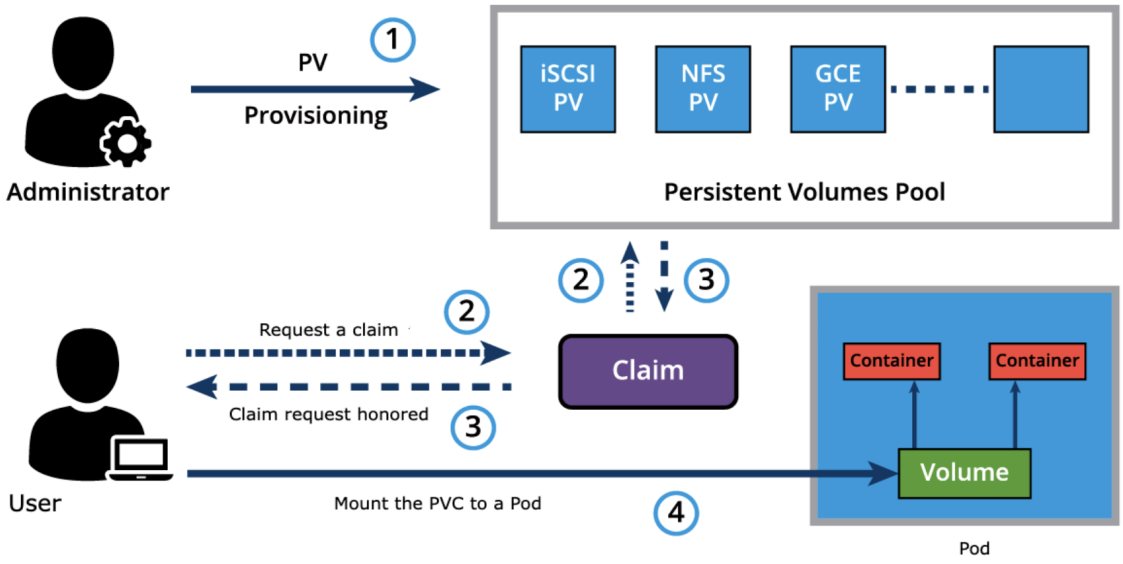
To manage the Volume, it uses the PersistentVolume API.
To consume it, it uses the PersistentVolumeClaim.
PV can be static provisioned by admin. Could be in the same place with the pods.
PC can be dynamic provisioned. StorageClass will define provisioner and volume details. User use PersistentVolumeClaims to send out request, the Volume is wired to StorageClass. (Need some example here)

PersistentVolumesClaim (PVC):

User send PersistentVolumeClaims request based on StorageClass + access mode (optional: + volume mode + file system + block device)

Access mode:

ReadWriteOnce (read-write by a single node)
ReadOnlyMany (read-only by many nodes)
ReadWriteMany (read-write by many nodes)
ReadWriteOncePod (read-write by a single pod).



Container Storage Interface (CSI):

For storage vendors, it is challenging to manage different Volume plugins for different container orchestrators (K8S, Mesos, Dockers,...).
CSI = standardize interface for Volume plugin

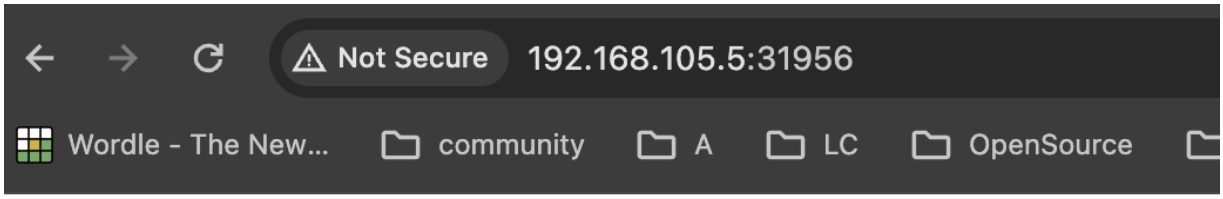
Demo of Shared hostPath Type Volume:

In the following sample, we create one pod with two containers and one shared volume (host-volume).
The volume is hostPath type and mounted at /home/docker/blue-shared-volume
Both container are mounting the host-volume so the volume is shared by both container
One of the container is running Debian image. It will create a index.html in /host-vol/ (this container's host-volume directory)

```
# app-blue-shared-vol.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: blue-app
  name: blue-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: blue-app
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: blue-app
        type: canary
    spec:
      volumes:
      - name: host-volume
        hostPath:
          path: /home/docker/blue-shared-volume
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: host-volume
      - image: debian
        name: debian
```

```
$ kubectl apply -f app-blue-shared-vol.yaml
deployment.apps/blue-app created

$ kubectl expose deployment blue-app --type=NodePort
service/blue-app exposed

$ kubectl get deploy,rs,ep,pod,svc -l app=blue-app
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/blue-app 1/1 1 1 12m

NAME DESIRED CURRENT READY AGE
replicaset.apps/blue-app-85db4f7f75 1 1 1 12m

NAME ENDPOINTS AGE
endpoints/blue-app 10.244.120.93:80 75s

NAME READY STATUS RESTARTS AGE
pod/blue-app-85db4f7f75-z66lh 2/2 Running 0 12m

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/blue-app NodePort 10.107.243.143 <none> 80:31956/TCP 75s

$ minikube service list
|----------------------|--------------------------|--------------|-----------------
-----------|
| NAMESPACE | NAME | TARGET PORT | URL |
|----------------------|--------------------------|--------------|-----------------
-----------|
| default | blue-app | 80 | http://192.168.105.5:31956 |
| default | kubernetes | No node port | |
|----------------------|--------------------------|--------------|-----------------
-----------|
```

When you open browser, you see the nginx is running on 192.168.105.5:31956.



Welcome to BLUE App!

This demo shows the two container are mounting the same volume. The Debian container did override the index.html of nginx.