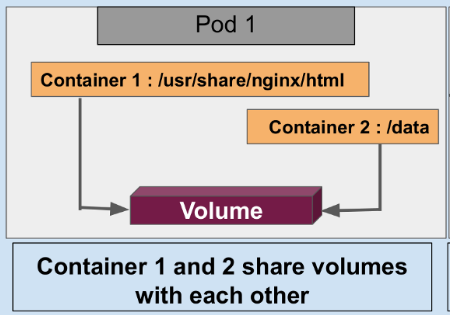
**Volume Resource in Kubernetes:**

**Ref Link: https://kubernetes.io/docs/concepts/storage/persistent-volumes/**

A single volume can assign to two or more containers in the same pod to share data. Volume lifecycle is depended on the pod in which it is put. It’s a place where the developer keeps the data for his application support it can be some part of coding.



A ***PersistentVolume* (PV)** is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using [Storage Classes](https://kubernetes.io/docs/concepts/storage/storage-classes/). It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes but have a lifecycle independent of any individual Pod that uses the PV.

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, see [AccessModes](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes)).

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

|  |  |
| --- | --- |
| **Access Modes** | A PV can have the following access modes:  ●      ReadWriteOnce—enables read and write and can be mounted by only one node  ●      ReadOnlyMany—enables read only and can be mounted by multiple nodes  ●      ReadWriteMany—both read and write, can be mounted by several nodes    Note: Different storage plugins support some of these access modes. |
| **Reclaim Policy** | The reclaim policy specifies what happens when the node no longer needs the persistent storage. It can be set to **Retain**, meaning the PV is kept alive until it is explicitly deleted; **Recycle**, meaning the data is scrubbed but can be restored later; and **Delete**, meaning it is irreversibly deleted.    Note: Different storage plugins support some of these reclamation policies. |

Install awscli on k8s master node:

<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-linux.html>

Configure AWS IAM to execute aws cli command from master node: -

apt-get install -y nfs-common

1. Create the AWS Elastic Block Store (EBS) volume in the *same region as your cluster*. If you have the aws cli installed and configured, this command will create one for you:

aws ec2 create-volume --availability-zone=eu-central-1c --size=10 --volume-type=gp2

2. With this new volume, attach it onto the master node in your cluster.

aws ec2 attach-volume --device /dev/xvdf --instance-id <MASTER NODE INSTACE ID> --volume-id <YOUR VOLUME ID>

3. In the master node, check to see if your device is attached to your instance by running lsblk. If the last step worked, you should see your volume at the bottom of the list. In this case, the volume I made earlier is called xvdf .

root@ip-172-31-1.2:~# lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

loop0 7:0 0 96.5M 1 loop /snap/core/9436

loop1 7:1 0 18M 1 loop /snap/amazon-ssm-agent/1566

loop2 7:2 0 28.1M 1 loop /snap/amazon-ssm-agent/2012

loop3 7:3 0 97M 1 loop /snap/core/9665

xvda 202:0 0 10G 0 disk

└─xvda1 202:1 0 10G 0 part /

xvdf 202:80 0 10G 0 disk

4. With the name of the volume, create the filesystem on the volume. This only needs to be done *once* on the volume.

sudo mkfs -t xfs /dev/<NAME OF VOLUME FROM PREV STEP>

5. Create a Persistent Volume that associates the EBS you made to the cluster. An example of said volume looks like this:

#### **pv.yaml**

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv

labels:

type: aws-pv

spec:

capacity:

storage: 3Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: <YOUR EBS VOLUME ID HERE>

fsType: xfs

kubectl create -f pv.yaml

6. Create the Persistent Volume Claim that will take a partition of the Persistent Volume we just made. An example of said claim would look like is:

#### **pvc.yaml**

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: aws-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 3Gi

7. Create a Pod that takes in the Persistent Volume Claim we just made and mounts it into the Pod. An example of said pod looks like this:

#### **Nginx-vol.yaml**

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- name: myfrontend

image: nginx

volumeMounts:

- mountPath: "/var/www/html"

name: mypd

volumes:

- name: aws-pv

persistentVolumeClaim:

claimName: aws-pvc

or

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- name: myfrontend

image: nginx

volumeMounts:

- mountPath: /var/www/html

name: aws-pv

volumes:

- name: aws-pv

awsElasticBlockStore:

volumeID: <volume-id>

fsType: ext4

8. Run the following kubectl commands on your cluster:

kubectl create -f pvc.yaml

To verify that your volume and claim are associated, run kubectl get pvc and look for the name of your PVC that you made.

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

aws-pvc Bound aws-pv 3Gi RWO 3s

If the status of it says BOUND, everything is working!

9. With you PVC bound to the PV, now run: kubectl create -f nginx-vol.yaml

10. Once it is up, verify to see if the volume has been properly mounted onto the pod by doing: kubectl describe pod mypod. If the Events section looks like the following, the volume mounted successfully!

11. Perform a local exec into the pod, using

kubectl exec -it <nameOfPod> -- /bin/bash

and verify that the volume is at the mount point that we specified (in this case, it should be at /var/www/html).

12. You’re done! Feel free to add files to that directory. Even if the pod is deleted, when the pod is respun up, whether it is the same exact yaml that we provided or if it is a brand-new pod, that file should still be in there.