STORAGE OVERVIEW [SECTION 42]

1. Container file System.
2. Volumes
3. Persistent Volumes
4. Volume Types.
5. Use K8s Volumes.
   a. Volumes & Volumes mounts
   b. hostpath & EmptyDir volumes
   c. Share Volume
6. Storage class.
7. Reclaim policy
8. Persistent Volume Claim

SECTION 42 : [267].

STORAGE OVERVIEW

In this section we will learn about Overview of K8s Storage.

1. Container file System
2. Volumes
3. Persistent Volumes
4. Volume Types.

Container file System :

1. Container file System is Ephemeral (Something that lasts of a very short time)
2. files in Container file System Exists only as long as the Container Exists.
3. Data in Container file System is lost as soon as Container Deleted or recreated.

So these kind of implementation is helpful, where you are working with the Stateless application where you don't need to maintain the State of your application.

But if you are working with the Stateful application where the application need to maintain the State of application & the data which is generated by the application is very necessary & Crucial part of your application, then this kind of implementation is <u>not worthy</u> because. The data which is generating within the application, within the Container that is being Removed as soon as the Container Removed.

② Volumes:

1) Many Application needs a Persistent Data (data need to Stored even if Container is deleted or recreated)

2) Volumes allows to Store Data Outside the Container, while allow Container to Access Data at Run Time.

You can Export the Container file System into the External Storage. & when it is like you can demand the Resources CPU & memory in the same way you can demand the Storage as well.

So a Storage will be assigned to the Container at <u>Runtime</u> & that External Storage is not dependent On the Container lifecycle (deleted or recreated).

3) Persistent Volumes:

1) Persistent Volumes are a bit more advanced than Volumes

2) Persistent Volumes allow user to treat Storage as an abstract resources & Consume it using pods.

it means like the pods and deployment, you can create the object or resource of ~~pre~~ Persistent Volume, in the K8s & that resource will be attached with the Container at runtime whenever you will define that resource in the persistent volume in the Container manifest

4) Volume Types:-

In K8s there is a Variety of the Volume Type & we need to understand the use of each Volume Type.

① Volume & Persistent Volumes each have a Volume Type:

These Volume must have a Type because that Volume Type will define how & where the Volume will be attached within your pods or Containers.

② Volume Type determines how Storage will be handled.

③ Various Volume Types Supports in K8s:

(a) NFS - N/w file System.
(b) Cloud Storage - AWS, GCP, Azure
(c) Config Maps & Secrets.
(d) File System on K8s Node.

[268]. use K8s Volume :-

will discuss use of Storage in K8s, how we can use that Storage in K8s.

1. Volumes & Volume Mounts
2. Sharing Volumes in Containers.
3. Common Volume Types.
4. Hands-on - Demo.

Volumes & Volume mount

1. Volume :- in pod Specification (→ manifest), user can define the Storage Volume available in for the pod.

   pod will use that Specified Storage.

2. Volume Specify the Volume Type and where the data is actually store.

   With in Volume Section (in manifest) you can define the Volume name, the Volume type & the other parameters. Regarding the Volume like size & retention policy.

3. Volume Mount :- Volume Mount is Container Specification (in manifest yml file), Refer the Volume in pod & Spec & provide a Mountpath.

Note: [Volume is pod Specification.
Volume Mount is Container Specification] of manifest /template / definition file.

```
apiversion: V1
Kind: Pod
metadata:
  name: pv-recycler
Spec:
  Volumes:
    - name: sample-vol
      hostpath:
        path: /data.
  Containers:
    - name: pv-recycler
      image: "k8s.gcr.io/busybox"
      Command: ["/bin/sh", "-c", "echo hello Team"]
      Volume Mounts:
        - name: sample-vol
          mountpath: /output
```

Pod Specification

Container Specif.

1) hostpath is Volume Type.

2)

Empty Dir Volume :-

1) EmptyDir: EmptyDir Created when pod is assigned to Node & Persist as long as pod Running on the Node.

So in the Earlier manifest we have seen we have defined the host path.

Similar to hostpath. you can also define the EmptyDir.

(a) K8s will Create a Volume dynamically whenever the pod will start, & that Volume will Only Exist untill & unless the pod is running.

(b) As Soon as you will delete the pod, the Same dynamic Volume will also deleted.

What is the benefit of Empty Dir.?

Cos it is following Same like Container file System procedure (deleted if pod is deleted).

(c) Benefits is:-

multiple Containers Can Mount the Same EmptyDir Volume

it means whenever, you are Creating the Empty Dir Volume within the pods. the Empty Dir Volume is Shareable that will be Shared b/w the multiple Containers, which is the part of that Particular pod.

(d) multiple Containers in the Pod Can read & write the Same files in the EmptyDir Volume, Though that Volume Can be mounted at the Same or different paths in each Container

It means within the same pod you can define the multiple Containers & all these Containers can refer the same empty Dir by this particular functionality.

All these Containers can refer to the same file system.

it means the file system is basically shared b/w the Containers & this approach has multiple benefits.

```
apiversion: V1
Kind: Pod
metadata:
  name: pv-recycler
Spec:
  Volumes:
    - names: Cache-vol.
      EmptyDir: {}          -> Volume type:
  Containers:
    - name: pv-recycler
      image: "K8s.gcr.io/busybox"
      Command: ["/bin/sh", "-c", "echo Hello Team"]
      VolumeMounts:
        - name: Cache-vol
          mountPath: /cache.
```

Share Volume :-

1) User can use the same VolumeMounts to share the same Volume to multiple Container within the same pod.

   that is EmptyDir volume Type.

2) this is very powerful feature which can be used to data transformation of Data Processing.

One use Case Example :-

Where over Container One generate the data, & Container two Can transform that data to some other form.

So you can we this kind of approach in the data processing, data mining & multiple business needs.

3) hostpath & empty Dir. Volume Type Support Share Volumes.

hostpath is also support Sharing Volumes.

```
apiVersion: v1
kind: Pod
metadata:
  name: pv-recycler
spec:
  Volumes:
  - name: Cache-Vol
    EmptyDir: {}
  Containers:
    - name: pv-recycler
      image: "k8s.gcr.io/busybox"
      Command: ["/bin/sh", "-c", "echo Hello Team"]
      VolumeMounts:
        - name: Cache-Vol
          mountPath: /Cache
  Containers:
    - name: pv-recycler-1
      image: "k8s-gcr-io/busybox"
      Command: ["/bin/sh", "-c", "echo Hello Team"]
```

```
VolumeMounts:
  - name: Cache-vol
    mountPath: /Cache/tmp.
```

[269] Lab:- Using Volume : Hostpath & EmptyDir.

manifest for pod with hostpath Volume Type.

```
apiVersion: V1
Kind: Pod.
metadata:
   name: hostpath-pod
Spec:
   Volumes:
    - name: hostpath-vol
      hostpath:
         path: /var/tmp.
   Containers:
    - name: hostpath-pod
      image: "k8s.gcr.io/busybox"
      Command: ["/bin/sh", "-c", "echo Hello Team"]
      VolumeMounts:
         - name: hostpath-vol.
         - mountPath: /output.
```

for Remaining Step & precedure Refer Lab Course.
or d. Person Desktop.

[270]. Lab Using Volume: Common Volumes.

```
apiVersion: v1
kind: Pod
metadata:
  name: shared-multi-container
spec:
  Volumes:
    - name: html
      emptyDir: {}
  Containers:
    - name: nginx-container
      image: nginx
      VolumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html
    - name: debian-container
      image: debian
      Volume Mounts:
        - name: html
          - MountPath: /html
      Command:
        - /bin/sh
        - '-c'
      args:
        - while true; do data >> /html/index.html;
                    sleep 5; done.
```

For Remaining Step & Procedure See lab Course or Personal desktop.

[27] :-

PERSISTENT VOLUMES in K8S.

1. Persistent Volumes.
2. Storage classes.
3. Persistent Volume claims.
4. Resizing Persistent Volume Claim.

Persistent Volumes :-

1. Persistent Volumes are K8s object that allow user to treat Storage as an Abstract Resource.

So last lecture/lesson we have seen the Volume, & VolumeMount as pod Specification and Container Specification.

But Persistent Volume is altogether K8s object. which is a K8s object like the pod, deployment, Services - etc.

* It will provide the Storage as a resource to the pods.

2. Persistent Volume is resources in the cluster just like a node is a cluster resource.

3. Persistent Volume uses a set of Attribute to describe the underlying Storage resources (Disk or, Cloud Storage), which will be used to store data.

with the help of the persistent Volume, you can define the Storage either on local disk. it means the K8s node disk or either on the Cloud disk.

There could be the multiple flavours available in the Persistent Volumes.
you can create the External disk with the multiple public Cloud providers.

Sample manifest for the Persistent Volume:-

```
apiVersion : V1
Kind : PersistentVolume  ->
metadata :
   name : Static-persistent-Volume
Spec:
  Capacity :
    Storage : 1 Gi
  accessModes :
    - ReadWriteMany
  hostpath :
    path : /Var/tmp
  StorageClassName : local-Storage
```

What is Storage Classes ?

1) StorageClass allows K8s Administrator to specify all type of Storage Service they offer on their platform.

you can Create the Persistent Volume with the multiple types, & you can Create it in the local disk of node or you can Create the Persistent Volume on a multiple Cloud Services.

But Somebody need to Control that where the user is Creating the Persistent Volume Right?

That is Control by Storage Class

The K8s administrator Can Specify all type of Storage Services, which is supported within your K8s cluster. & that thing is done by the K8 Storage class,

definition / manifest.

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

1) Storage class is another K8s object
2) when you define no provision means this Storage class will be provisioned on your local machine.
3] Volume Binding mode : WaitForFirstConsumer means - whenever this particular volume, the Storage class is Created it will bind with the pod.

So here we are defining wait for first Consumer Once the first pod will be Created. & refer to this. Particular Storage class,

Once that particular pod will be created only this particular storage block will not be reserved.

2) In similar way we can define another kind of storage class.

(a) Admin cloud create a storageclass called Slow to describe inexpensive storage for general development use.

your development team wants to create the multiple pods they want to create the multiple instances of your application & your application have a need of the external storage then in the case the community administrator need to make sure that the storage will be available & inexpensive.

So you can create a storage called Slow or something like that. & you can create that particular class on a cloud storage.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

(b) Admin cloud Create a Storage Class Called FAST for High I/O Operation Applications. (For GCP)

```
apiVersion: Storage.k8s.io/v1
Kind: StorageClass.
metadata:
    name: fast
provisioner: Kubernetes.io/gce-pd.
parameters:
    type: pd-ssd.
allowed Topologies:
- matchLabel Expressions:
    - Key: failure-domain.beta.kubernetes.io/zone
      Values:
        - us-central1-a
```

3) with in the Storage Class you will get an option allow Volume Expansion :-

(1) allow Volume Expansion :- This field can accept boolean Value only.

(2) This is the property of StorageClass & defines whether StorageClass supports the ability to Resize after they are Created.

(3) All cloud disk Supports this property.

```
apiversion: Storage.k8s.io/v1.
Kind: StorageClass.
metadata:
    name: local-Storage
provisioner: Kubernetes.io/no-provisioner
VolumeBindingMode : WaitForFirstConsumer.
allowVolumeExpansion: true.
```

Reclaim policy :-

(1) Persistent Volume Reclaim Policy :- This define, how the Storage will be treated, when the Persistent Volume associated PVC are deleted.

persistent Volume claim

Volume claim which will attach with the persistent Volume.

So what happened? Storage data?

Once the persistent ~~with the Storage~~ Volume claims are deleted

(2) There could be three Option's.

(a) Retain :- Keep all the data, This require manual data cleanup & prepare for reuse

(b) Delete :- Delete underlying Storage Resources automatically (Support for Cloud Resource Only)

it means Once the persistent Volume Claims are deleted, it will delete the underlying Storage Resources as well.

(c) Recycle :- Automatically delete all data in underlying Storage, Allow persistent Volume to be reuse.

```
apiVersion : v1
kind : PersistentVolume
metadata :
  name : Static-persistent-volume
spec :
  capacity :
    storage : 1Gi
accessModes :
  - ReadWriteMany
hostPath :
  path : /var/tmp .
StorageClassName : local-storage .
PersistentVolumeReclaimPolicy : Recycle .
```

Persistent Volume claim :

① Persistent Volume claim (PVC) is a request for storage by a user.

So whenever a user is basically trying to attach the Persistent Volume with the pod, with the resources they can request that particular Persistent Volume by a Persistent Volume claims.

without Persistent Volume ch. claim, you cannot attach the persistent volume with your resources either Container or pods.

② Persistent Volume claims :- define a set of attribute similar to those of PVs. (Persistent volumes).

③ PVCs look for a PVs that is able to meet the criteria if it found one, will automatically be bound to that PV.

```
apiVersion: v1
kind: PersistentVolumeClaim.        -> object.
metadata:
  name: myclaim.
Spec:
  accessModes:
    - ReadWriteMany.
  resources:
    requests:
      Storage: 1Gi
StorageClassName: local-storage.
```

Once you will attach this particular persistent volume claim, with your pod first, what it will do is, it will try to identify the persistent volume which is matching with the same specification which is defined for this particular persistent volume claim.

if it will find out the persistent volume, it will attest that persistent volume with the resource.

Otherwise (if not found) it will create altogether a New Persistant Volume dynamically & attach that persistent volume with the resources.