<https://blog.couchbase.com/stateful-containers-kubernetes-amazon-ebs/>

<https://blog.couchbase.com/kubernetes-cluster-amazon-expose-service/>

<https://github.com/kubernetes-incubator/external-storage/tree/master/aws/efs>

<https://www.juandebravo.com/2018/09/28/aws-efs-kubernetes/>

https://containerjournal.com/topics/container-networking/using-ebs-and-efs-as-persistent-volume-in-kubernetes/

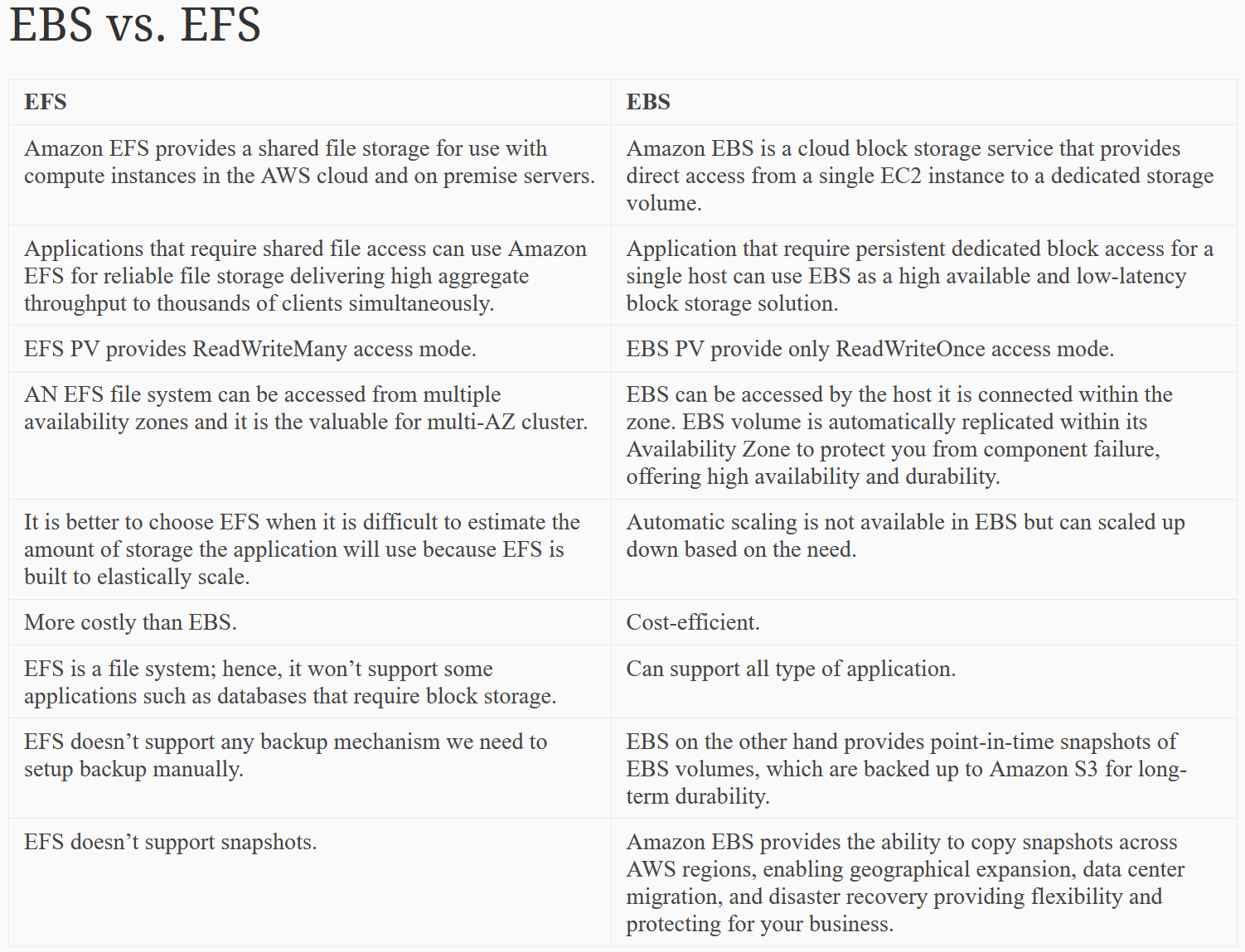
<https://medium.com/@while1eq1/using-amazon-efs-in-a-multiaz-kubernetes-setup-57922e032776>

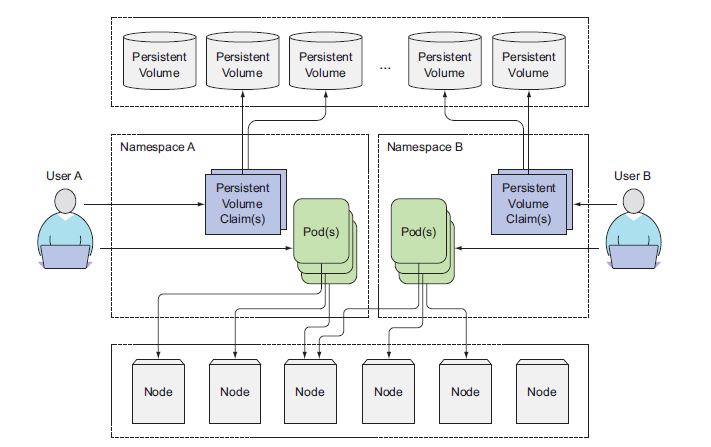
<https://docs.giantswarm.io/guides/using-persistent-volumes-on-aws-with-efs/>

<https://github.com/kubernetes-incubator/external-storage/blob/master/aws/efs/deploy/manifest.yaml>

<https://rancher.com/running-highly-available-wordpress-mysql-kubernetes>

<https://medium.com/swlh/stupid-simple-kubernetes-persistent-volumes-explained-by-examples-29f8fec08c4>





===================EMPTY-DIR===========================================

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- name: mypod1

image: sreeharshav/rollingupdate:v1

volumeMounts:

- name: testvol

mountPath: /tmp/testvolume

volumes:

- name: testvol

emptyDir: {}

============================HOSTPATH-VOL=========================================

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- name: mypod1

image: sreeharshav/rollingupdate:v1

volumeMounts:

- name: testvol

mountPath: /tmp/testvolume

- name: mypod2

image: sreeharshav/utils

volumeMounts:

- name: testvol

mountPath: /tmp/data

volumes:

- name: testvol

hostPath:

path: /tmp/

type: Directory

nodeSelector:

kubernetes.io/hostname: ip-172-20-52-2.ec2.internal

MOUNTING RAW EBS DISK TO POD WITHOUT PV & PVC:

1.Create a EBS Volume in the AZ where u want to deploy the POD.

2. Add Tag to the EBS VOlume as below.

Tag Name: KubernetesCluster Value=<YourClusterName>

Eg: EBS Tags => KubernetesCluster: trainingk8s.xyz

<https://stackoverflow.com/questions/47278433/need-help-on-volume-mount-issue-with-kubernetes>

3. Use below manifest for POD & Deployment and make sure you change the hostname under the node selector as per your cluster.

4. If you get below error when you perform kubectl describe pod, reboot the node once.



==================POD-RAW-EBS-MAPPING========================

apiVersion: v1

kind: Pod

metadata:

name: test-ebs

labels:

name: raw-volume-testing

spec:

containers:

- image: mongo

name: mongodb

imagePullPolicy: Always

volumeMounts:

- name: mongodb-data

mountPath: /data/db

volumes:

- name: mongodb-data

awsElasticBlockStore:

volumeID: vol-0755d708d698d035a

fsType: ext4

nodeSelector:

kubernetes.io/hostname: ip-172-20-76-92.ec2.internal

=================DEPLOYMENT-WITH-RAW-DEVICE=================

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongo-deployment

labels:

app: mongodb

spec:

replicas: 1

selector:

matchLabels:

app: mongodb

name: mongodb

template:

metadata:

labels:

app: mongodb

name: mongodb

spec:

containers:

- image: mongo

name: mongodb

imagePullPolicy: Always

volumeMounts:

- name: mongodb-data

mountPath: /data/db

volumes:

- name: mongodb-data

awsElasticBlockStore:

volumeID: vol-0755d708d698d035a

fsType: ext4

nodeSelector:

kubernetes.io/hostname: ip-172-20-76-92.ec2.internal

4. Exec in to the container and type mongo to open the mongoshell.

5. Execute following commands to create the DB and create test data.

> use k8sclass #This will create a db called k8sclass

db.movie.insert({"name":"Fast & Furious 1"})

db.movie.insert({"name":"Fast & Furious 2"})

db.movie.insert({"name":"Fast & Furious 3"})

db.movie.insert({"name":"Fast & Furious 4"})

db.movie.insert({"name":"Fast & Furious 5"})

db.movie.insert({"name":"Fast & Furious 6"})

> show dbs

> show collections

> db.movie.find()

{ "\_id" : ObjectId("5fba07002b3305220b382302"), "name" : "Fast & Furious 1" }

{ "\_id" : ObjectId("5fba07002b3305220b382303"), "name" : "Fast & Furious 2" }

{ "\_id" : ObjectId("5fba07002b3305220b382304"), "name" : "Fast & Furious 3" }

{ "\_id" : ObjectId("5fba07002b3305220b382305"), "name" : "Fast & Furious 4" }

{ "\_id" : ObjectId("5fba07002b3305220b382306"), "name" : "Fast & Furious 5" }

{ "\_id" : ObjectId("5fba07022b3305220b382307"), "name" : "Fast & Furious 6" }

EBS is a persistent block storage with a defined size (It is however possible to change it later).

It is fully configurable and could be the most performant solution of Amazon.

The major drawback is that it can be only mounted to one EC2 instance.

This also means that it can't be shared between multiple pods.

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

<https://blog.abyssale.com/shared-storage-volume-on-amazon/>

Mapping a RAW Disk

a. EBS Volume is available in only one AZ.

b. Developers need knowledge on AWS to create vol or vol-id.

c. No data protection in case of AZ Failure.

d. Since EBS is also not shared between the EC2 VM, multiple PODs cannot

write at the same time.

========================================================================

PV & PVC:

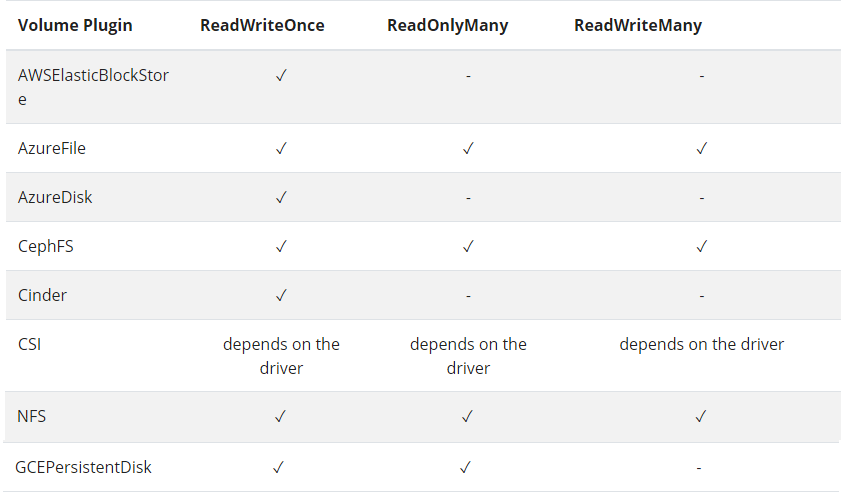
* RWO - ReadWriteOnce
* ROX - ReadOnlyMany
* RWX - ReadWriteMany

The access modes are:

* ReadWriteOnce -- the volume can be mounted as read-write by a single node(AWS EBS)
* ReadOnlyMany -- the volume can be mounted read-only by many nodes
* ReadWriteMany -- the volume can be mounted as read-write by many nodes( AWS EFS)

A volume will be in one of the following phases:

* Available -- a free resource that is not yet bound to a claim
* Bound -- the volume is bound to a claim
* Released -- the claim has been deleted, but the resource is not yet reclaimed by the cluster
* Failed -- the volume has failed its automatic reclamation



Give following tags to the PV, otherwise deletion of PVC wont result in EBS Deletion.

Tag Name: KubernetesCluster Value=<YourClusterName>

Eg: EBS Tags => KubernetesCluster: trainingk8s.xyz

----------------PV---------------------

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv1

labels:

type: aws-pv1

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-04180df2e5277b4ea

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv2

labels:

type: aws-pv2

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-04180df2e5277b4ea

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv3

labels:

type: aws-pv3

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-04180df2e5277b4ea

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv4

labels:

type: aws-pv4

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-04180df2e5277b4ea

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv5

labels:

type: aws-pv5

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-04180df2e5277b4ea

fsType: ext4

------------PVC----------------------

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim1

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim2

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim3

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim4

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim5

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

----------------PV---------------------

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv1

labels:

type: aws-pv1

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-0eb77fa6ddf3c1d79

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv2

labels:

type: aws-pv2

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-0eb77fa6ddf3c1d79

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv3

labels:

type: aws-pv3

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-0eb77fa6ddf3c1d79

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv4

labels:

type: aws-pv4

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-0eb77fa6ddf3c1d79

fsType: ext4

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: aws-pv5

labels:

type: aws-pv5

spec:

storageClassName: gp2

persistentVolumeReclaimPolicy: Delete

capacity:

storage: 2Gi

accessModes:

- ReadWriteOnce

awsElasticBlockStore:

volumeID: vol-0eb77fa6ddf3c1d79

fsType: ext4

#------------PVC----------------------

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim1

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim2

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim3

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim4

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: task-pv-claim5

spec:

storageClassName: gp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

------------POD-WITH-PVC--------------------------

apiVersion: v1

kind: Pod

metadata:

name: task-pv-pod

spec:

volumes:

- name: task-pv-storage

persistentVolumeClaim:

claimName: task-pv-claim1

containers:

- name: task-pv-container

image: nginx

ports:

- containerPort: 80

name: "http-server"

volumeMounts:

- mountPath: "/usr/share/nginx/html"

name: task-pv-storage

------------DEPLOYMENT-WITH-PVC--------------------------

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongo-deployment

labels:

app: mongodb

spec:

replicas: 1

selector:

matchLabels:

app: mongodb

name: mongodb

template:

metadata:

labels:

app: mongodb

name: mongodb

spec:

containers:

- image: mongo

name: mongodb

imagePullPolicy: Always

volumeMounts:

- name: mongodb-data1

mountPath: /data/db

- name: mongodb-data2

mountPath: /tmp/db1

- name: mongodb-data3

mountPath: /tmp/db2

- name: mongodb-data4

mountPath: /tmp/db3

volumes:

- name: mongodb-data1

persistentVolumeClaim:

claimName: task-pv-claim1

- name: mongodb-data2

persistentVolumeClaim:

claimName: task-pv-claim2

- name: mongodb-data3

persistentVolumeClaim:

claimName: task-pv-claim3

- name: mongodb-data4

persistentVolumeClaim:

claimName: task-pv-claim4

nodeSelector:

kubernetes.io/hostname: ip-172-20-37-221.ec2.internal

-----------------------------------------------------------------

Reclaiming

When a user is done with their volume, they can delete the PVC objects from the API that allows reclamation of the resource. The reclaim policy for a PersistentVolume tells the cluster what to do with the volume after it has been released of its claim. Currently, volumes can either be Retained, Recycled, or Deleted.

Retain

The Retain reclaim policy allows for manual reclamation of the resource. When the PersistentVolumeClaim is deleted, the PersistentVolume still exists and the volume is considered "released". But it is not yet available for another claim because the previous claimant's data remains on the volume.

kubectl patch pv aws-pv5 -p '{"spec":{"claimRef": null}}'

for I in {1..6}

do

kubectl patch pv aws-pv$I -p '{"spec":{"claimRef": null}}'

done

Delete

For volume plugins that support the Delete reclaim policy, deletion removes both the PersistentVolume object from Kubernetes, as well as the associated storage asset in the external infrastructure, such as an AWS EBS, GCE PD, Azure Disk, or Cinder volume. Volumes that were dynamically provisioned inherit the reclaim policy of their StorageClass, which defaults to Delete.

Recycle

Warning: The Recycle reclaim policy is deprecated. Instead, the recommended approach is to use dynamic provisioning.

RETAIN STORAGE CLASS:

---

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: awsgp2retain

provisioner: kubernetes.io/aws-ebs

reclaimPolicy: Retain

parameters:

zone: us-east-1a

#zones: us-east-1a, us-east-1b

type: gp2

fsType: ext4

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: auto-assign-pv-retain

spec:

storageClassName: awsgp2retain

accessModes:

- ReadWriteOnce

resources:

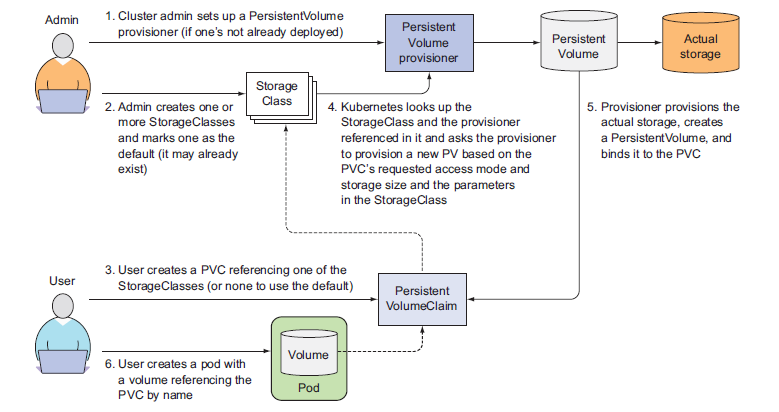
requests:

storage: 2Gi

kubectl patch pv pvc-aeb04fd2-49ab-487a-aa89-1cfd6c868f85 -p '{"spec":{"claimRef": null}}'

DYNAMIC-STORAGE-PROVISIONING:

Dynamic volume provisioning allows storage volumes to be created on-demand. Without dynamic provisioning, cluster administrators have to manually make calls to their cloud or storage provider to create new storage volumes, and then create [PersistentVolume objects](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) to represent them in Kubernetes. The dynamic provisioning feature eliminates the need for cluster administrators to pre-provision storage. Instead, it automatically provisions storage when it is requested by users.



===============DYNAMIC-PROVISIONING====================

#kubectl api-resources | grep -i storage

#kubectl api-versions | grep -i storage

#kubectl explain storageclasses

#kubectl get storageclass

---

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: awsgp2

provisioner: kubernetes.io/aws-ebs

parameters:

type: gp2

fsType: ext4

---

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: awsio1

provisioner: kubernetes.io/aws-ebs

parameters:

type: io1

#zone: us-east-1a

zones: us-east-1a, us-east-1b

iopsPerGB: "5"

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: auto-assign-pv-1

spec:

storageClassName: high

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 4Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: auto-assign-pv-2

spec:

storageClassName: medium

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

DYNAMIC PROVISIONING WITH PVC:

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: auto-assign-pv-1

spec:

storageClassName: awsgp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 4Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: auto-assign-pv-2

spec:

storageClassName: awsio1

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 4Gi

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: auto-assign-pv-3

spec:

storageClassName: awsgp2

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 2Gi

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongo-deployment

labels:

app: mongodb

spec:

replicas: 1

selector:

matchLabels:

app: mongodb

name: mongodb

template:

metadata:

labels:

app: mongodb

name: mongodb

spec:

containers:

- image: mongo

name: mongodb

imagePullPolicy: Always

volumeMounts:

- name: mongodb-data-1

mountPath: /data/db

- name: mongodb-data-2

mountPath: /data/db2

- name: mongodb-data-3

mountPath: /data/db3

volumes:

- name: mongodb-data-1

persistentVolumeClaim:

claimName: auto-assign-pv-1

- name: mongodb-data-2

persistentVolumeClaim:

claimName: auto-assign-pv-2

- name: mongodb-data-3

persistentVolumeClaim:

claimName: auto-assign-pv-3

nodeSelector:

kubernetes.io/hostname: ip-172-20-33-116.ec2.internal

db.helo.insertMany([

{ "\_id" : 1, "name" : "Matt", "status": "active", "level": 12, "score":202},

{ "\_id" : 2, "name" : "Frank", "status": "inactive", "level": 2, "score":9},

{ "\_id" : 3, "name" : "Karen", "status": "active", "level": 7, "score":87},

{ "\_id" : 4, "name" : "Katie", "status": "active", "level": 3, "score":27, "status": "married", "emp": "yes", "kids": 3}

])

db.helo.find({name: "Katie"})

db.helo.find()