vol types

i) node local type such as ~~emptydir~~ empty dir and hostpath.

ii) File sharing such as NFS

iii) Cloud provider, AWS EBS or Azure disk

iv) distributed file systems glusterfs or cephfs

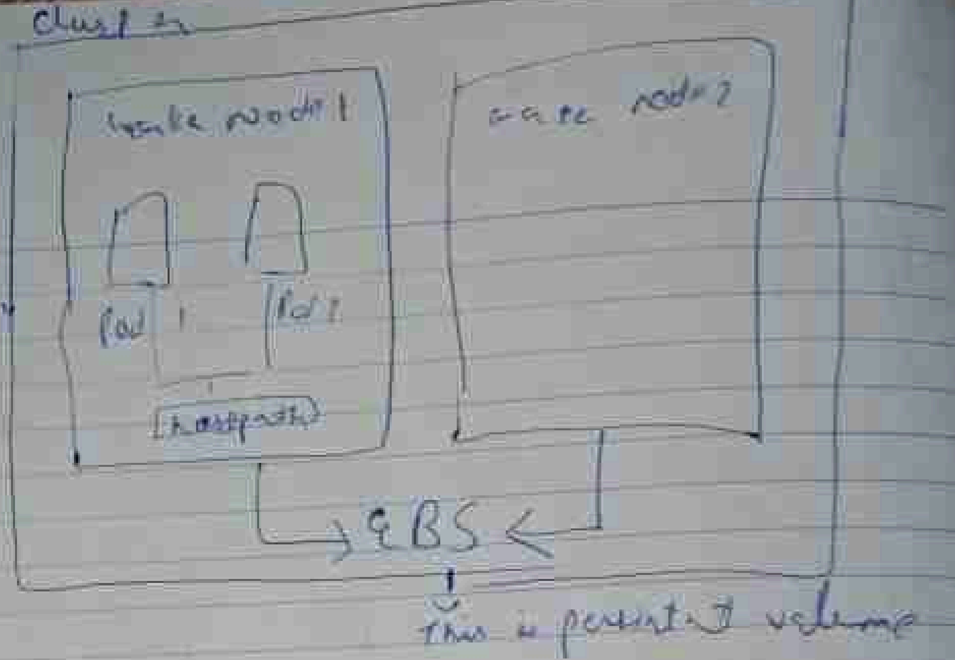
v) Special like secret, git repo

* PERSISTENT VOLUME AND LIVENESS PROBE

A persistent volume (PV) is a cluster-wide resource that you can use to store data in a way that it persists beyond the lifetime of a pod.

The PV is not backed by locally attached storage on a worker node but by networked storage system such as EBS or NFS or a distributed file system like ceph.

K8s provides APIs for users and administrators to manage and consume storage. To manage the volume, it uses the persistent volume API resource type and to consume it, uses the persistent volume-claim API resource type



PV Claim

- The PVC request a PV with your desired specification (size, access speed / from EBS) and once PV is found, it is bond TO PV Claim.
- After a successful bound to a pod, you can mount it as a volume.
- Once a user finishing it work, the attached PV volume can be released. The underlying PV can there be reclaimed and recycled for future usage.

Restrictions using AWS EBS

1) The nodes on which pods are running must be AWS EC2 instance.

2) The instances need to be in same region and Availability zone as EBS volume

3) EBS only support a single EC2 instance mounting a volumes

Health Check / Liveness Probe

- Health checks or probes are carried out by the kubelet to determine when to ~~restart~~ recreate a container (for liveness probe) and used by service and deployment to determine if a pod should receive traffic.

- One use of readiness probe is to control which pod are used as backends for service. When a pod is not ready, it is removed from service load balancer.

initial delay second = time do container start hone ke liye
period seconds = agla health check kab karna hai
timeout seconds = kitni baar checking karna aur container restart karo

Configmap AND SECRETS

Configmap can be used in following ways :-

1) As env variable
2) As vol in the pod

```
Kubectl create configmap <mapname> --from-file
 = <file to read>
```

SECRETS

1) Secrets are namespaced objects, that is exist in the context of a namespace.

2) The secret data on nodes is stored in tmpfs volumes. tmpfs is a file system which keeps all file in virtual ~~[illegible]~~ memory. Everything in tmpfs is temporary in sense that no files will be created on your hard drive.

3) You can access via env variable or volume

4) A per secret size limit of 1 MB exists

5) The API server stores secret as plaintext in etcd

* Secrets can be created

1) from a text file
2) from a yaml file.

(To make a secret)

- kubectl create secret generic mysecret --from-file=username.txt --from-file=pwd.txt

- kubectl get secret

1 CPU = 1000 m

memory : mi B

x NAMESPACE , Limits and Requests

- A namespace is a group of related element that each have a unique name or identifier. Namespace is used to uniquely identify one or more names from other similar names of different objects, groups or the namespace in general

- A scope for every names

- A mechanism to attach authorization and policy to a subsection of the cluster.

- Most K8s resources (pods , services , replication controllers) ~~and~~ are in same namespace and low ~~so~~ level resources such as nodes and persistent volumes are not in any namespace.

* To change from default namespace,
```
kubectl config set-context $(kubectl config current-context) --namespace = dev
```

* To create a pod in your namespace
```
kubectl apply -f pod.yaml -n dev ← namespace name
```

* Managing Compute Resources :—
  - Scheduler decides about which nodes to place pods, only if the node has enough CPU resources available to satisfy the pod CPU request

* Two type of Constraints Can be set of each resource type
1) Request and 2) Limits

1) A request is the amount of that resource that the system will guarantee for the container and K8s will use this value to decide on which node to place the pod.

2) A limit is the max amount of resources that K8s will allow the container to use. In the case that request is not set for a container, it default to limit if limit is not set, then it default to 0. (unlimited)

Request = not mention	Request = mention
Limit = ~~mention~~ mention	Limit = no mention
Request = limit	limit = default

You can limit
1) Compute, memory, storage.

2 Restrictions
1) Every Container that runs on namespace must have its own CPU limit
2) Total amount of CPU used by all container in the namespace must not exceed a specified limit.

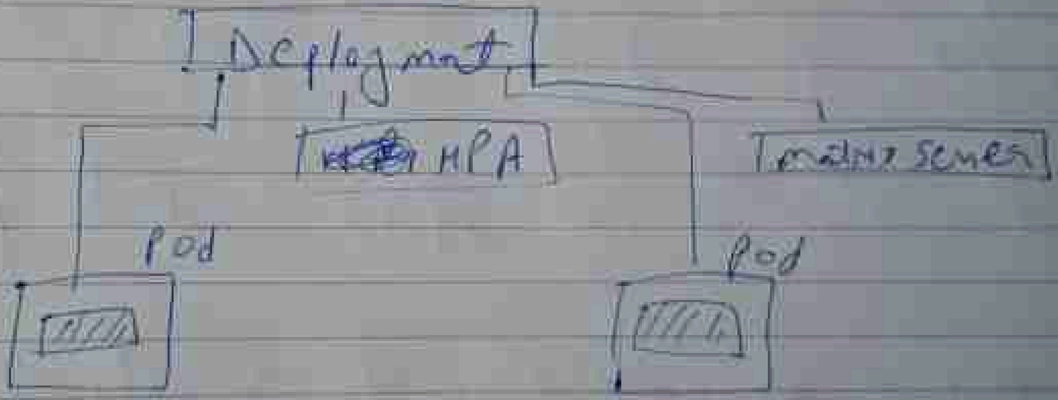
Resource Quota and Horizontal Scaling

Default Range →

CPU	Memory
(min) request - 0.5	(min) request - 500 M
(max) limit - 1	(max) limit - 1G

- Horizontal Pod Autoscaler.

- K8s has the possibility to automatically scale pods based on observed CPU



cooling period = 5 min
HPA check in every 30 second

- Scaling can be done only for scalable objects like controller, deployment or Replica set

- HPA is implemented as a K8s API resource and a controlle

- HPA is implemented as a controlled loop with a period controlled by the controller manager.

horizontal pod autoscaler sync period

Jobs

Jobs gets stop after doing his work.

Objects like replicaset, daemonsets, statefullset and deployments they will make ensure their pod are running

Cron Job
If we have multiple nodes hosting the applica for high availability, which nodes handle cron?

* Init Container.
- Init Containers are specialized Containers that run before app containers in a pod.
- Init containers always run to completion
- Init container do not support readiness probe.

Use case,
1) Clone a git repository into a volume.
2) Generate configuration file dynamically.

* POD LIFECYCLE

Pending	Running	Succeded	failed	Completed	Unknown

1) Pending.
- The pod has been accepted by the K8s system but it's not running.
- If the pod cannot be scheduled because of resource constraints

2) Running

- The pod has been bound to a node.
- All containers have been created

3) Succeeded

All containers in the pod have terminated in success and will not be restarted.

4) Failed

All container in the pod terminated and atleast one container has terminated in failure.

The container either exited with non-zero status or was terminated by system

5) Unknown

- state of the pod could not be obtained
- Typically due to error in network or communicating with the host of the pod

6) Completed

The pod has run to completion as there's nothing to keep it running

POD Conditions

Kubectl describe pod <Podname>

1) PodScheduled : Pod has been scheduled

2) Ready : added to load balancing pool

3) Initialized : All init containers have started successfully

4) Unscheduled : can't schedule right now

5) ContainersReady : All containers are Ready