KUBERNETES- K8s

A container management tool

KUBERNETES-Introduction

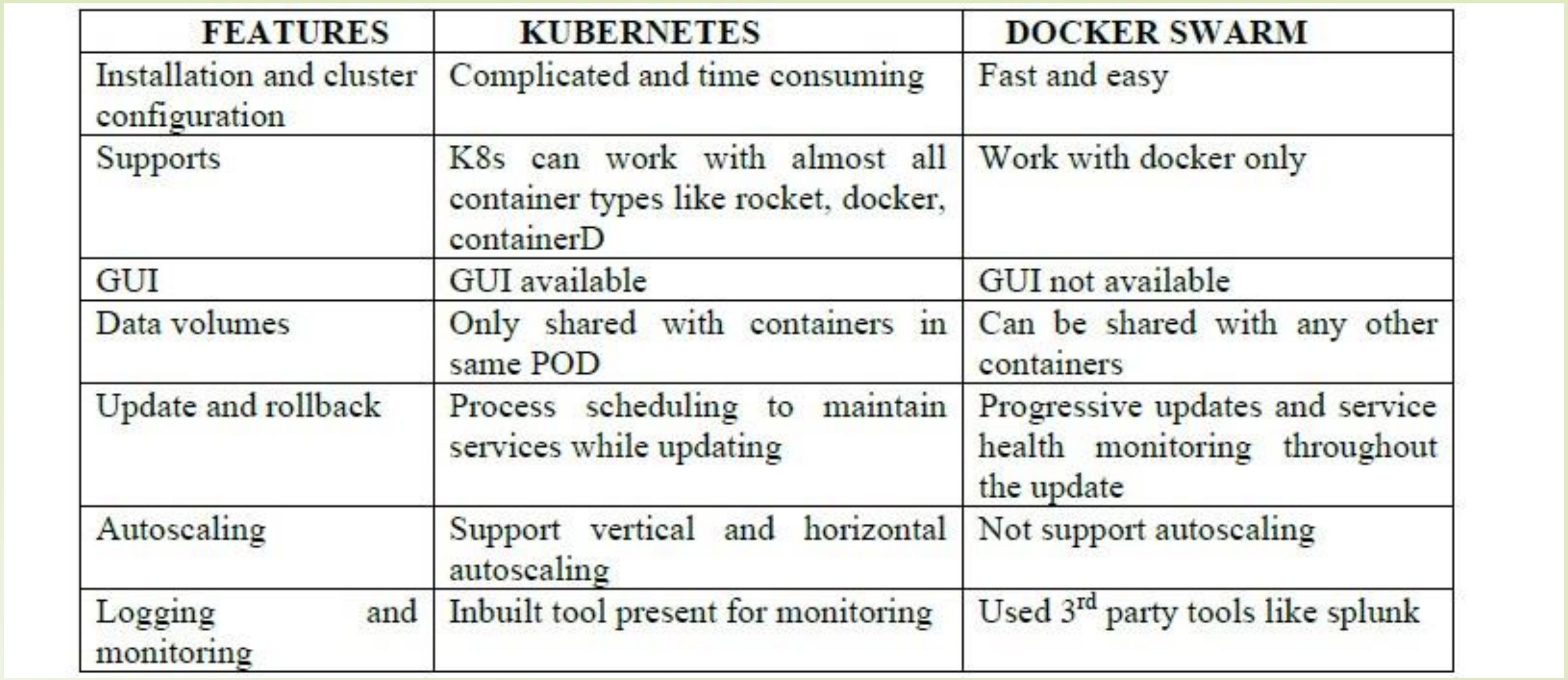
- ➢ Kubernetes is an open-source container management tool which automates container deployment, container scaling and load balancing.
- ➢ It schedules, runs and manages isolated containers which are running on virtual/ physical/ cloud machines.
- ➢ All top cloud providers support Kubernetes.
- History:
- ➢ Google developed an internal system called 'borg' (later named as omega) to deploy and manage thousands google application and services on their cluster.
- ➢ In 2014, google introduced Kubernetes as an open-source platform written in Golang, and later donated to CNCF (cloud Native Computing Foundation).
- Online platform for K8s:
- i. Kubernetes playground
- ii. Play with K8s
- iii. Play with Kubernetes classroom

KUBERNETES-Intro

- Cloud based K8s services:
- i. GKE: Google Kubernetes Services
- ii. AKS: Azure Kubernetes services
- iii. Amazon EKS: Amazon Elastic Kubernetes Services
- Kubernetes installation tool:
- i. Minicube
- ii. Kubeadm
- Problems with scaling up the containers:
- ➢ Containers cannot communicate with each other.
- ➢ Autoscaling and load balancing was not possible.
- ➢ Containers had to be managed carefully

Features of Kubernetes:

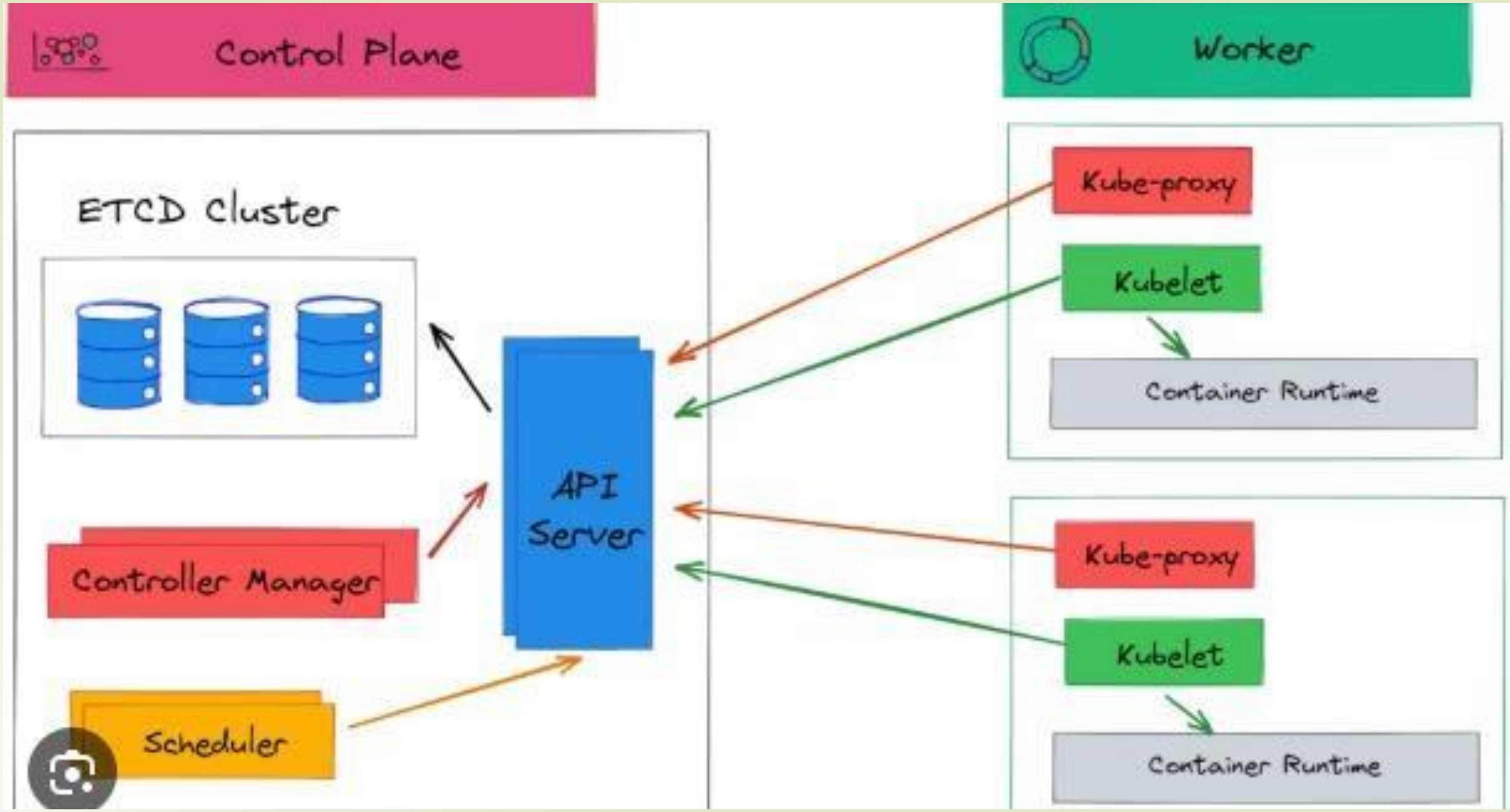
- ➢ Orchestration (clustering of any number of containers running on different networks)
- ➢ Autoscaling (supports both horizontal and vertical scaling)
- ➢ Auto-healing
- ➢ Load balancing
- ➢ Platform independent (cloud/ virtual/ physical)
- ➢ Fault tolerance (node/ POD failure)
- ➢ Rollback (going back to previous version)
- ➢ Health monitoring of containers
- ➢ Batch execution (one time, sequential, parellel)



FEATURES	KUBERNETES	DOCKER SWARM
Installation and cluster configuration	Complicated and time consuming	Fast and easy
Supports	K8s can work with almost all container types like rocket, docker, containerD	Work with docker only
GUI	GUI available	GUI not available
Data volumes	Only shared with containers in same POD	Can be shared with any other containers
Update and rollback	Process scheduling to maintain services while updating	Progressive updates and service health monitoring throughout the update
Autoscaling	Support vertical and horizontal autoscaling	Not support autoscaling
Logging and monitoring	Inbuilt tool present for monitoring	Used 3rd party tools like splunk

Kubernetes and Docker-Swarm Comparison

Kubernetes-Better option



K8'S

KUBERNETES- ARCHITECTURE

Working with Kubernetes

- ➢ We create manifest (.yml/json).
- ➢ Apply this to cluster (to master) to bring desired state.
- ➢ POD runs on node which is controlled by master.
- Role of Master Node:
- ➢ Kubernetes cluster contains containers running or bare metal/ vm instances. Cloud instances/ all mix.
- ➢ Kubernetes designates one or more of these as master and all others as workers.
- ➢ The master is now going to run set of k8s processes. These processes will resume smooth functioning of cluster. These processes are called "control plane".
- ➢ Can be multi-master for high availability.
- ➢ Master runs control plane to run cluster smoothly.
- **Components of Control Plane (master):**
- 1. Kube-API server
- 2. Etcd
- 3. Kube-scheduler
- 4. Controller manager

Components of Control Plane (master):

- 1. Kube-API server (for all communications):
- ➢ This api-server interacts directly with user (i.e we apply .yml or json manifest to kube-apiserver).
- ➢ This kube-apiserver is meant to scale automatically as per load.
- ➢ Kube api-server is front-end of control-plane.
- 2. Etcd:
- ➢ It stores metadata and status of cluster.
- ➢ Etcd is consistent and high available store (key-value store).
- ➢ Source of touch for cluster state (info about state of cluster).
- Etcd has following features: -
- a. Fully replicated: the entire state is available on every node in the cluster.
- b. Secure: implements automatic TLS with optional client-certificate authentication.
- c. Fast: benchmarked at 10,000 writes per second.

Components of Control Plane (master):

- 3. Kube scheduler:
- ➢ When users make request for the creation and management of PODs, kube scheduler is going to take action on these requests.
- ➢ Handles POD creation and management.
- ➢ Kube scheduler match/ assign any node to create and run PODs.
- ➢ A scheduler watches for newly created PODs that have no node assigned. For every POD that the scheduler discovers the scheduler becomes responsible for finding best node for that POD to run on.
- ➢ Scheduler gets the information for hardware configuration from configuration file and schedules the PODs on nodes accordingly
- 4. Controller Manager:
- ➢ Make sure that actual state of cluster matches the desired state.
- ➢ Two possible choices for controller manager:
- a. If k8s on cloud, then it will be cloud-controller-manager.
- b. If k8s on non-cloud then it will be kube-controller-manager

Components on master that runs controller:

- a. **Node controller:** for checking the cloud providers to determine of a node has been detected in the cloud after id stops responding.
- b. **Route controller:** responsible for setting up network, routes on your cloud.
- c. **Service controller:** responsible for load balancers on your cloud against services of type load balancers.
- d. **Volume controller**: for creating, attaching and mounting volumes and interacting with the cloud provider to orchestrate volume.

Nodes (kublet and container engine.)

- Node is going to run 3 important pieces of software/ process.
- **1. Kublet**
- **2. Container engine**
- **3. Kubeproxy**
- 1. Kublet:
- ➢ Agent running on the node.
- ➢ Listens to Kubernetes master. (e.g-POD creation request)
- ➢ Use port 10255.
- ➢ Send success/fail reports to master.
- 2. Container Engine:
- ➢ Works with kublet.
- ➢ Pilling images.
- ➢ Start/ stop containers.
- ➢ Exposing containers on ports specified in manifest.
- 3. Kubeproxy:
- ➢ Assign IP to each POD.
- ➢ It is required to assign IP address to PODs (dymanic).
- ➢ Kube-proxy runs on each node and this make sure that each POD will get its own unique IP address. These 3 components collectively called NODE.

POD:

- ➢ Smallest unit in Kubernetes.
- ➢ POD is a group of one or more containers that are deployed together on the same host.
- ➢ A cluster is a group of nodes.
- ➢ A cluster has at least one worker node and master node.
- ➢ In Kubernetes the control unit is the POD, not containers.
- ➢ It consists of one or more tightly coupled containers.
- ➢ POD runs on node which is control by master.
- ➢ Kubernetes only knows about PODs (does not know about individual container).
- ➢ Cannot start containers without a POD.
- ➢ One POD usually contains one container.
- Multi container PODs:
- ➢ Share access to memory space.
- ➢ Connect to each other using local host <container port>.
- ➢ Share access to the same volume.
- ➢ Containers within POD are deployed in an all-or-nothing manner.
- ➢ Entire POD is hosted on the same node (scheduler will decide about which node).

POD

- POD limitations:
- ➢ No auto healing or auto scaling.
- ➢ POD crashes.
- **Higher level Kubernetes objects:**
- a. Replication set: auto scaling and auto healing
- b. Deployment: versioning and rollback
- c. Service:
- d. Static (non-ephemeral) IP and networking.
- e. Volume: non-ephemeral storage.
- Important:
- Kubectl- single cloud
- Kubeadm- on premise
- Kubefed- federated

Kubernetes Objects

- Anything which has shape and size is called Object.
- In Kubernetes, Work is called Objects e.g. in Manifest file we write, this is our desired state, we want this. It's called Object.
  - Object can be Container creation, Open port etc.
- Kubernetes uses object to represent state of the cluster,Which Containerized applications are running and on which node.
- Policies around how these applications behave such as restart policies, upgrades and fault tolerance.
- Once object is created, Kubernetes system ensures Object exits and maintains.
- Every K8s objects includes two nested that govern object config.
- 1-The Spec- Describes the desired state.
- 2-The Status-Actual state of object ,supplied and updated by K8s System

Basic K8s Objects

- 1-Pod
- 2-Replicaset
- 3-Deployment
- 4-Service
- 5-Volumes
- 6-Configmaps
- 7-Secrets
- 8-Namespaces
- 9-Jobs
- 10-Daemon Sets

Relationship Between Objects

- Pod manages Containers.
- Replicaset manages Pods.
- Deployment Help Pod to Rollback to previous state.
- Services Expose Pod to outside world(expose to Internet)
- Configmaps and Secrets help to configure Pods.
- K8s objects are created by us, and we push them to K8s API through Kubectl.
- Kubectl is a command line Tool which supports several different ways to create and manage K8s Objects.
- Commands can be written in either Declarative or Imperative method.
- Declarative method used in Prod projects stating what trying to achieve without telling how to.
- Imperative method used in Dev Projects explicitly telling how to accomplish.

POD-Features

- When a POD is created, its scheduled to run in our cluster.
- POD remains in that Node until
- A-Process is terminated.
- B-POD object is deleted.,
- C-POD is evicted for lack of resources.,
- D-Node fails.
- If Node dies, PODS scheduled to that node are scheduled for deletion after timeout period.
- A given POD is not rescheduled to a new node , instead replaced with a new node even with same name but different UID.
- Volume in a pod will exist until Pod exists , if that POD is deleted, Volume also destroyed.
- A controller can create multiple Pods , manage ,handle replication, rollout provide self healing capabilities.

K8s Configurations

- All In One Single Node Installation:-

With All In One, both Master and Worker are installed on single node. Useful for Learning and Development not to be used for Production.

Minikube is one such example.

Single Node etcd,Single Master and multi-worker installation:-

In this we have single master node which runs on single etcd instance ,multiple workers are connected to Master node.

Single node etcd,Multi Master and multi worker- Installation:-

24*7 Availability,here we have multiple masters for HA(high availability)mode but we have a single node etcd instance. Multiple workers are connected to Master.

====To create a pod==================

- Start Minikube
- $ apt install conntrack
- $ minikube start --vm-driver=none
- $ minikube status

vi pod.yml

- kind: Pod ---------------------------- →Object of type POD
- apiVersion: v1
- metadata:
- name: testpod--→name of the pod
- spec--- →what is desired is written under spec
- containers:
- ----------- name: c00
- image: ubuntu
- command: ["/bin/bash", "-c", "while true; do echo Hello-Ashok; sleep 5 ; done"]
- ----------- name: c01
- image: httpd
- ports:
- --------------containerPort: 80

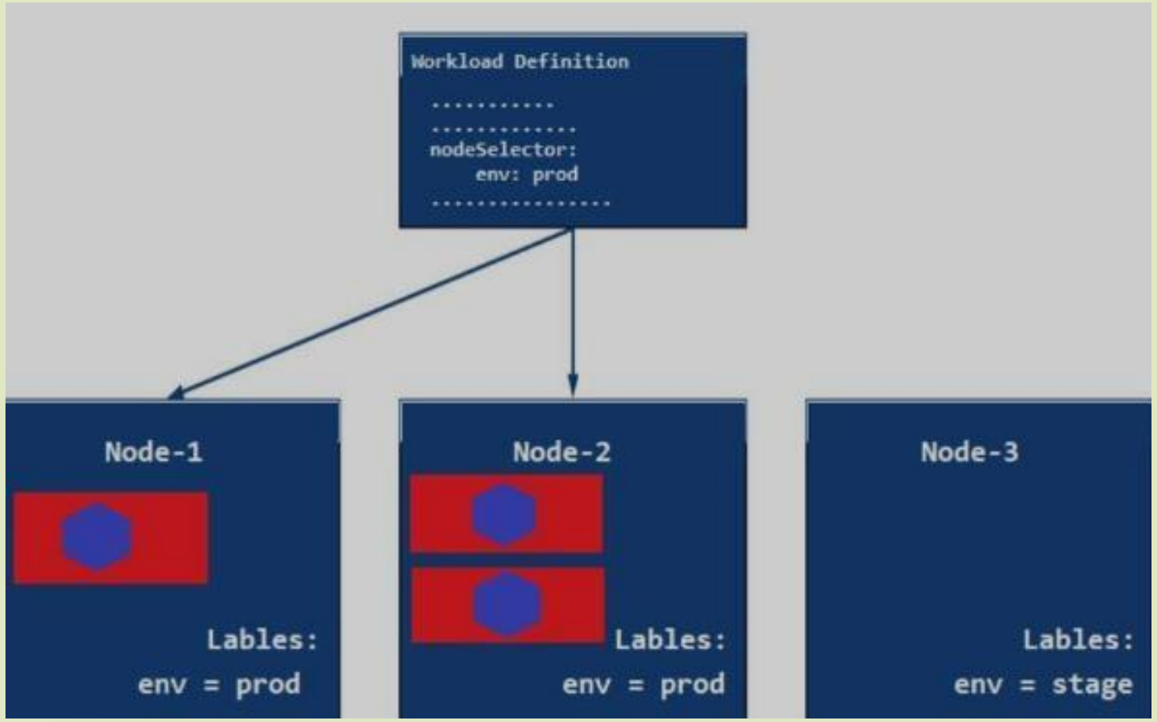
K8s Commands To Create Pod

- Kubectl apply –f pod.yml
- o/p --- pod/testpod created
- Kubectl get pods
- o/p testpod 1/1 running
- Kubectl describe pod testpod
- Above to see where running
- Kubectl logs –f testpod
- Kubectl get pods –o wide –To See details
- If required to delete pod
- Kubectl delete –f pod1.yml
- For Multi-container,
- Kubectl logs –f testpod3--→error
- Kubectl logs –f testpod3 –c coo1
- Kubectl exec testpod3 –it –c c01 –/bin/bash

Labels, Selectors, Replication controller,Replica Set and Node Controller

- Labels are the mechanism to organise K8’s objects.
- A label is key-value pair without pre-defined meaning that can be attached to objects.
- So we are free to choose labels to refer environment- Dev or Prod or test.
- Multiple labels can be attached to single objects.
- Vi pods.yml

```
kind:pod
apiversion:V1
Metadata:
    name:delhiprod
    labels:
        env: development
        class: pods
        Name: Ashok
        company: CBA
```



K8s Label Commands

- kubectl get pods - - show -labels
- Now,if we want to add label to existing pod.
- kubectl label pods delhiprod myname=Ashok
- Now,list pods matching a label
- kubectl get pods –l env=development
- Now list pods where label is not development
- kubectl get pods –l env!= development
- Now ,if we want to delete using labels
- kubectl delete pod –l env!= development
- Kubectl get pods

Selectors

- Unlike name/UIDs ,Lables do not provide uniqueness as in general,we can expect many objects to carry same label.
- Once labels are attached to an object, we may need filters to narrow down and these are called label-selectors.
- Api currently supports two types of selectors:-
- Equality based and Set-based
- Equality based= C= ,!=
- The Label selector is made up of multiple requirements which are comma separated.
- Set-based(in,not in,exists)
- Key→env in(Dev,prod)
- Key→env not in(team1,team2)
- K8s also supports set based selectors
- Match multiple values
- Kubectl get pods –l 'env in(development,testing)
- Kubectl get pods –l class=pods,myname=Ashok

Node-Selector

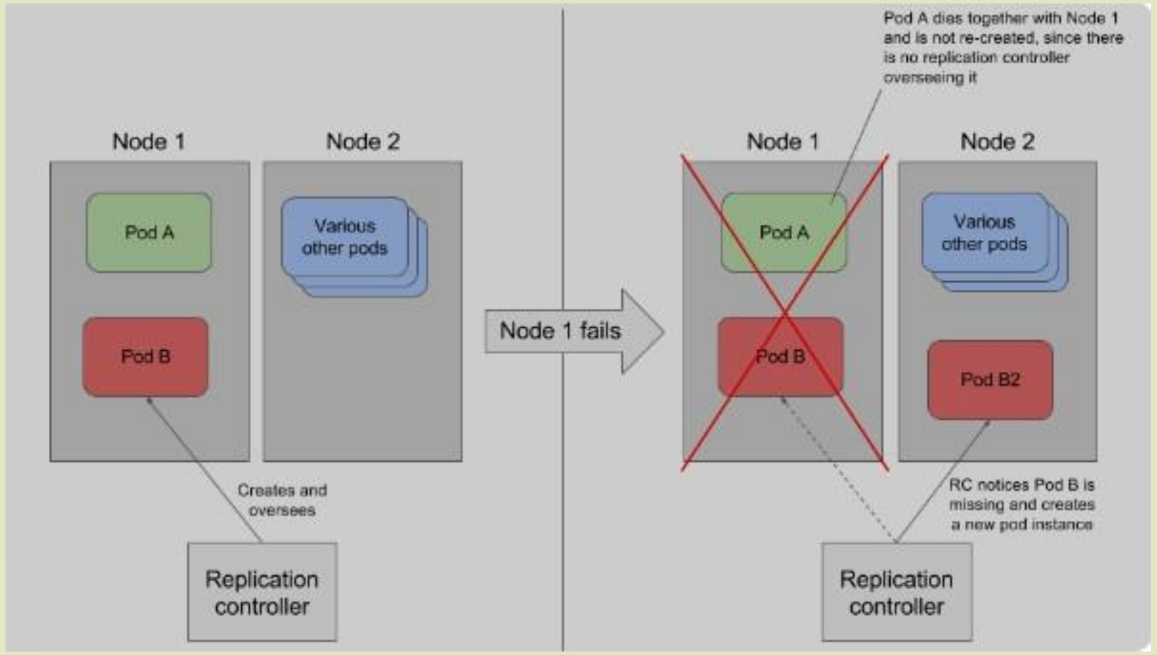
- If we want pod should be created on specific node e. g t2.medium,specifically mark hardware =t2.medium.
- If all are t2.medium,wherever master finds resources are more available, it creates there.
- One use case for selecting labels is to contain set of nodes onto which pod can schedule,we can tell POD to run on a particular node.
- Generally, such constraints are unnecessary as scheduler will automatically do reasonable placement but sometimes, we may need.
- We can use labels to tag nodes.
- Once nodes are tagged ,we can use label-selectors to specify Pods to run only on specific nodes.
- First, we give label to nodes.
- Then we use node selector for Pod configuration.

*Scaling and Replication(HA,24*7)*

- Kubernetes is designed to orchestrate multiple containers and Replication.
- Need for multiple containers/replication helps us with these
- **Reliability**→ By having multiple versions of an application, we can prevent problems ,if one or more fails.
- **Load-Balancing** → Having multiple versions of containers enables to easily send traffic to different instances to prevent over loading in single instances.
- **Scaling**→ when load becomes too much for the number of existing instances, Kubernetes enables to easily scale up application adding instances as needed.
- **Rolling updates**→ updates to a service by replacing pods one by one.
- Replicas 2→ 2 ditto pods created
- Replicas 3→3 ditto pods created

Replication Controller

- **A Replication controller is an object which enables us to create multiple pods to make sure that desired number of pods always exist.**
- **A pod created by Replication controller will be automatically replaced by a new pod in case pods get crashed, failed or terminated.**
- **RC is recommended if we want to make sure atleast 1 pod always exist even after system reboots.**
- **We can run the RC with 1 replica and RC will make sure the POD is always running.**


Node 1
Node 2
Pod A
Pod B
Various other pods
Creates and oversees
Replication controller
Node 1 fails
Pod A dies together with Node 1 and is not re-created, since there is no replication controller overseeing it
Node 1
Node 2
Pod A
Pod B
Various other pods
Pod B2
RC notices Pod B is missing and creates a new pod instance
Replication controller

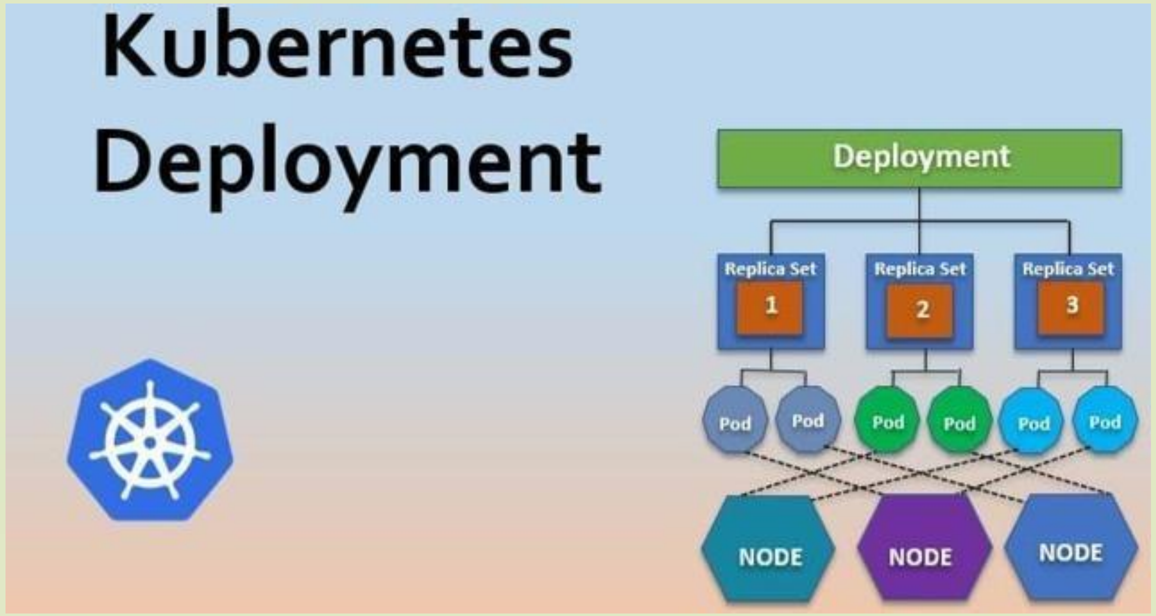
Replication Controller

- kind:Replication Controller----------→defines to create object of type RC
- apiversion: v1
- metadata:
-     name: myreplicas--------------- →Name of replica
- spec:----------------------------------- →What we desire
-     replicas:3------------------------------→no of desired pods

```
  selector: ------------------------------ → RC gets POD created on that node
     myname: Ashok-------------------→must match labels
  template: ---------------------------- →defines template to launch POD
     metadata:
        name: testpod
        labels: Ashok
  spec:
     containers:
        name: C00
        image: ubuntu
        command: ["/bin/bash", "-c", "while true; do echo Hello-Ashok; sleep 5 ; done"]
```

Deployment and Roll-Out and Rollback

- How to come back to previous state?
- A deployment object gives capability and control over how and when a new POD created can be rolled out, updated or Rolled back to previous state.
- Deployment is upper-level object than RS and PODS.
- When using Deployment object, we first define state of the app then K8s cluster schedules that app on specific node.
- K8s then monitors, if node hosting an instance goes down or POD is deleted then Deployment replaces providing self healing mechanism



Use Cases of Deployment

- Create a Deployment to roll out a replica set.
- Declare a new state of pod →By updating the Pod template spec of deployment a new replica set is created and the deployment manages moving the pods from old replica set to new one at a controlled rate.Each new Replica set updates the revision of deployment.
- Rollback to an earlier Deployment revision
- Scale up the Deployment to facilitate more load.
- Pause the Deployment to apply multiple fixes to its Pod template spec and then resume
- Clean up older Replica set that's not required any more.
- We can roll back to a specific version- by specifying it with - -to-revision
- Kubectl rollout undo deploy/mydeployment – to- revision=2
- The name of replicaset is always formatted as [Deployment-name]-[Random string]

Inspection of Deployment in Cluster

- When we inspect Deployment in our cluster and give below command
- kubectl get deploy
- Following Fields will appear
- Name→List the name of Deployment in namespaces
- Ready→Display how many replicas of application are available to your users.
- Up-To-Date→Display the number of replicas that have been updated to achieve the desired state.
- Available →Displays how many replicas of application are available to users
- Age→Display the amount of time application was running.

Deployment-Example

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name:  mydeployments
spec:
  replicas: 1
  selector:     # tells the controller which pods to watch/belong to
   matchLabels:
    name: deployment
  template:
    metadata:
      name: testpod1
      labels:
        name: deployment
    spec:
     containers:
       - name: c00
         image: httpd
         ports:
         - containerPort: 80
```

Deployment Commands

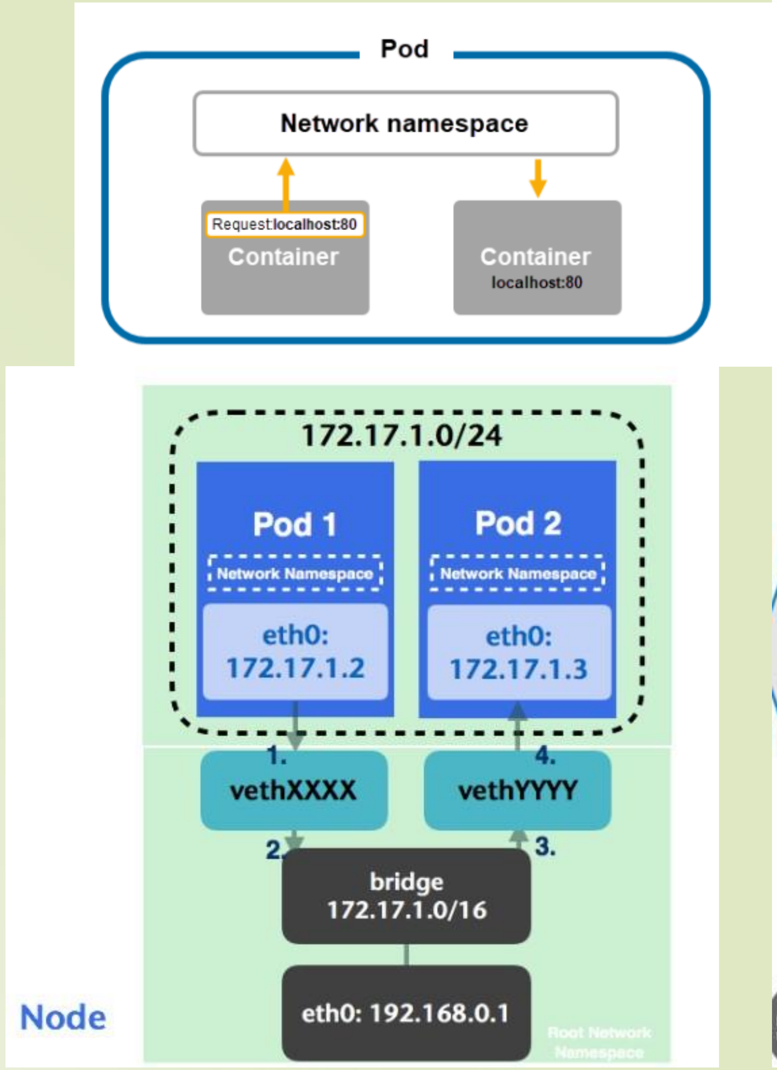
- To Check Deployment was created or not.
- kubectl get deploy
- To check how deployment creates RS and PODS
- kubectl describe deploy mydeployments
- kubectl get rs
- To Scale up or down
- kubectl scale - - replicas=1 deploy mydeployments
- To scale up or down
- kubectl scale - -replicas=1 deploy mydeployment
- To check what is running inside container
- Kubectl logs –F <podname>
- Kubectl rollout status deployment mydeployments
- Kubectl rollout history deployment mydeployments
- Kubectl rollout undo deploy mydeployments

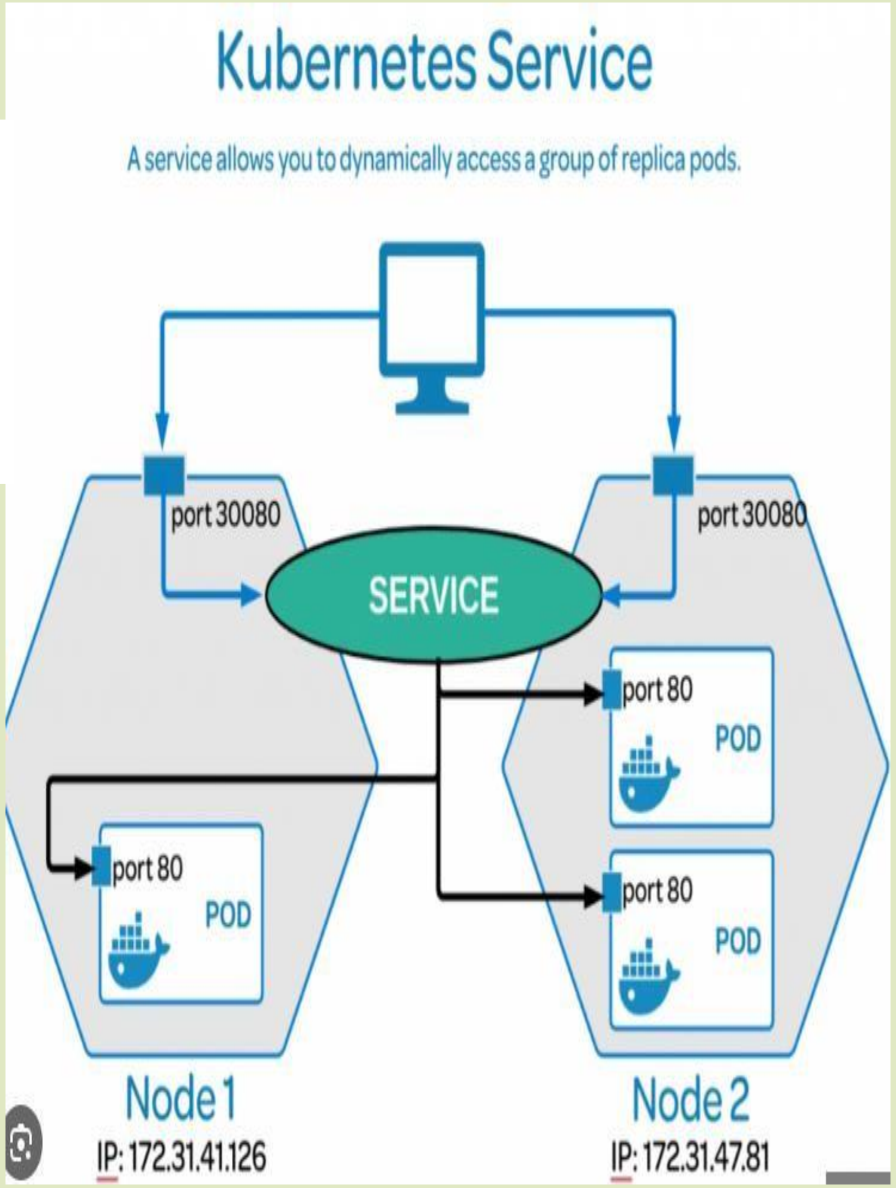
Failed Deployments

- Deployment may get struck trying to deploy its newest replicaset without completing.
- 1→Insufficient Quota
- 2→Readiness probe failures
- 3→Image pull errors
- 4→Insufficient permissions
- 5→Limit Ranges
- 6→Application run time misconfiguration

K8s Networking, Services, Node port

- Kubernetes Networking addresses four major concerns.
- 1→Containiner to Container within a pod use networking to communicate through loopback(localhost).
- 2→Cluster Networking provides communication between different pods
- 3→The service object lets to expose app running on pod to be reachable from outside world.
- 4→We can also use services to publish services only for consumption inside cluster.





K8s Networking, Services

- 1-Container to Container communication happens through localhost-within the container.
- Pod To Pod Communication on same worker node happens through POD IP.
- By default , POD IP will not be accessible from outside world.
- Object Services applied above RS to provide Virtual IP.
- Each POD gets its own IP Address.
- However, in deployment, the set of PODS running at one moment could be different from set of PODS running moment later, similarly RS creates new POD to maintain minimum value 1,so PODS can be different at different moment.
- This results in problem, if some set of PODS(call them 'Backend') provide functionality to other PODS(call them 'frontend'), inside cluster, how frontend pods keep a track of backend pods, through which IP ,they can contact.

K8s Networking, Services

- When using RS ,PODS are terminated and created during scaling or replication operations.
- Similarly ,while using Deployments, while updating image version, PODS are terminated, and new PODS take place of other PODS.
- So, PODS are very dynamic in nature they come and go and get created on nay node ,so difficult to access them as IP changes every time.
- Service objects are logical bridge between pods and end users which provides virtual IP(VIP)
- The VIP is not Actual IP connected to a N/W but its purpose is to forward traffic to one or more pods.
- Kube-proxy keeps mapping between VIP and POD up-to-date with queries to API servers to learn about new services in the cluster.
- Although each POD has unique IP but they are not exposed to outside world.
- Services help to expose the VIP mapped to the ports and allows application to receive traffic.

K8s Networking, Services

- Labels are used to select which are the PODS to be put under a service.
- Creating a service will create an end point to access the PODS/applications in it.
- Services can be exposed in many ways by specifying type in service spec.
- 1→Cluster IP(default)
- 2→Nodeport
- 3→LoadBalancer created by cloud providers they will route external traffic to every node(e.g. ELB),by default services can run between ports(30,000-32767),the set of pods target determined by selector.
- Cluster-IP exposes VIP only reachable from within the cluster.
- Mainly used to communicate between the components of micro-services
- Nodeport is applied above Cluster-IP and and Load-Balancer service typeis on top.
- Nodeport makes a service accessible from outside cluster
- Exposes services on the same port of each selected Node in cluster using NAT.Node gets mapped to VIP,if AWS EC2 has public DNS,Nodeport between(30,000-32,767) will be given.

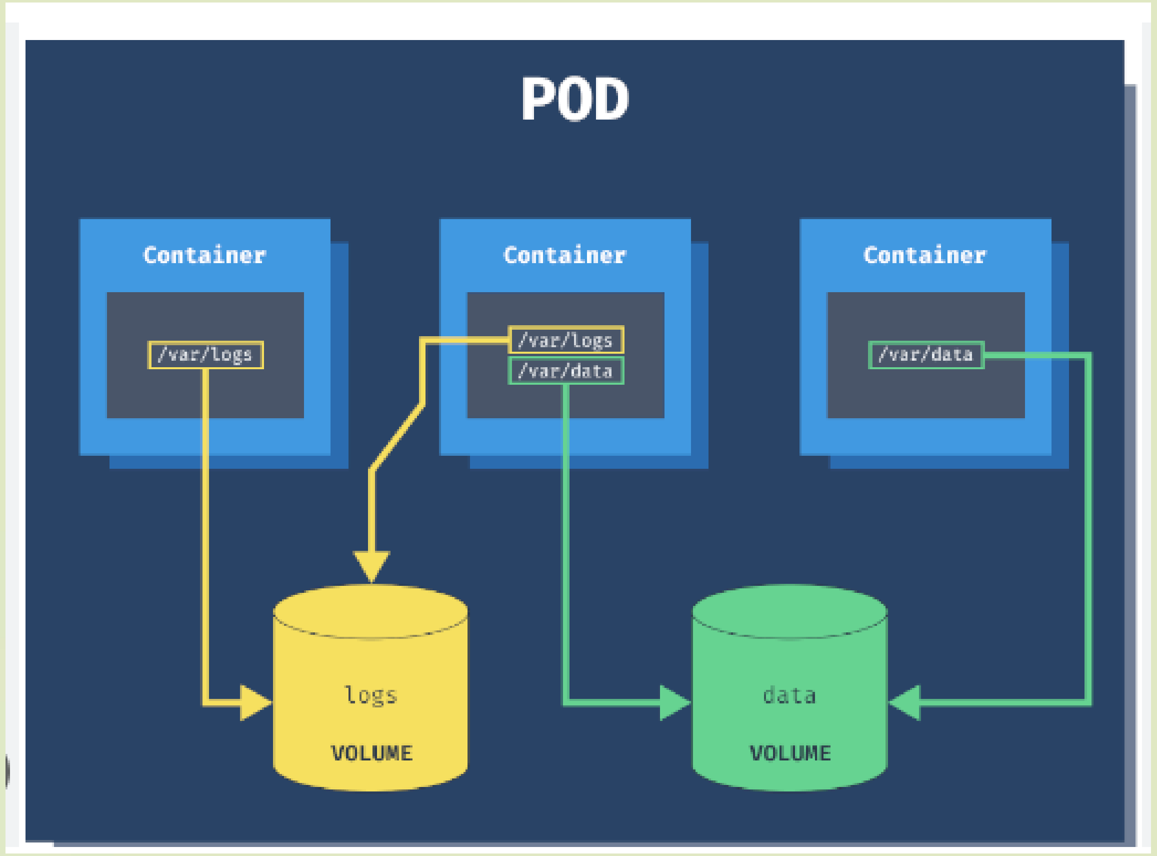
K8s Services-Lab

```
kind: Service                          # Defines to create Service type Object
apiVersion: v1
metadata:
    name: demoservice
spec:
    ports:
      - port: 80                       # Containers port exposed
        targetPort: 80                 # Pods port
    selector:
      name: deployment                 # Apply this service to any pods which has the specific label
    type: ClusterIP                    # Specifies the service type i.e ClusterIP or NodePort
```

```
$ kubectl get svc
```

Volumes

- Containers are short lived in nature.
- All data stored inside a container is deleted ,if container crashes. however, Kubernetes system will restart it , which means no old data.
- To overcome this,K8s uses volume. A volume is a directory backed by storage medium and its content is determined by volume type.
- In K8s volume is attached to a pod and shared among the containers of that POD.
- The Volume has the same life span as of POD and it outlives that of a container ,this allows the data to be preserved in case of container restarts.
- A Volume type decides the property of a directory like size , content etc e.g.
- Node local type such as emptydir and host path.
- Filesharing types like nfs
- Cloud provider specific types like AWS EBS,Azure etc.
- Distributed file system types e.g. glusternfs or cephfs.
- Special purpose types like secret,gitrepo



volume labs

```
apiVersion: v1
kind: Pod
metadata:
  name: myvolemptydir
spec:
  containers:
  - name: c1
    image: centos
    command: ["/bin/bash", "-c", "sleep 15000"]
    volumeMounts:                                    # Mount definition inside the container
      - name: xchange
        mountPath: "/tmp/xchange"
  - name: c2
    image: centos
    command: ["/bin/bash", "-c", "sleep 10000"]
    volumeMounts:
      - name: xchange
        mountPath: "/tmp/data"
  volumes:
  - name: xchange
    emptyDir: {}
```

EmptyDir

- We use this when we want to share content among different containers of same POD and not to host ,machine.
- An EmptyDir is first created when a POD is assigned to a node and exists as long as POD is running on that node.
  - AS name suggests, its empty initially.
- Containers in the POD can all read and write the same files in the emptyDir volume ,although that volume can be mounted at the same and different path in each containers.
- When the POD is removed from a node for any reason , data mounted on EmptyDir is also removed forever.
- Container crashes does not remove POD ,hence data in emptyDir is safe across container crashes.

HostPath

- We use this when we want to access the content of POD/Container from host machine.
- A hostpath volume mounts a file or directory from host nodes filesystem into your POD.
- Create a volume and map it to hostpath

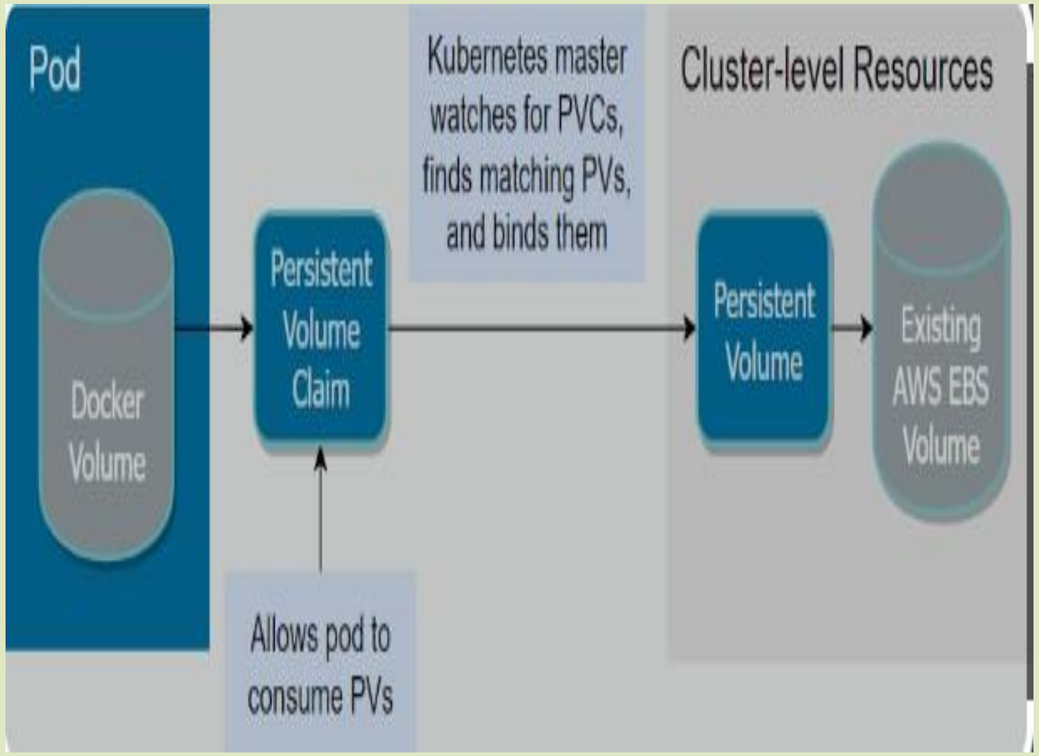
```
apiVersion: v1
kind: Pod
metadata:
  name: myvolhostpath
spec:
  containers:
  - image: centos
    name: testc
    command: ["/bin/bash", "-c", "sleep 15000"]
    volumeMounts:
    - mountPath:  /tmp/hostpath
      name: testvolume
  volumes:
  - name: testvolume
    hostPath:
      path: /tmp/data
```

Persistent Volume and Liveness Probe

- In IT Sector , Storage is managed by Storage admin not by users.
- In containerized world,K8s resolves the problem of handling Volume types with help of Persistent Volume(PV) subsystem.
- PV is not backed by locally attached storage on a worker node but by Network attached storage system such as EBS or EFS or distributed file system like Ceph.
- K8s provides API for users and administartors to manage and consume resources.
- To manage the volume, it uses the Persistent volume API resource type.
- And to consume it,uses the persistent –volume-claim API resource type.
- Normally,life of a volume exits till life of POD,if POD dies,RS,Deployment automatically creates a new POD in any node and volume associated with old POD ,goes with it.
- Solution is connect the two nodes with EBS and mount the path of POD to EBS to get volume shared even if POD dies.

Persistent Volume Claim

- In order to use a PV ,we need to claim it first using Persistent Volume claim.
- The PVC request a PV with desired specification(size,access modes,speed etc) and once a suitable persistent volume is found,it is bound to persistent volume claim.
- After a successful bound to a POD , we can mount it as a volume.
- Once a user finishes its work , the attached Persistent Volume can be released, the underlying PV can be reclaimed and recycled for future usage.
- AWS EBS Volume mounts EBS Volume into our POD unlike emptyDir contents of EBS Volume are preserved and volume is merely unmounted,Instance storage is non-persistent.
- There are a few restrictions.
- 1→Same region and Availability Zone.
- 2→EBS only Supports a single EC2 instance mounting a volume.



PERSISTENT VOLUME

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: myebsvol
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  awsElasticBlockStore:
    volumeID:        # YAHAN APNI EBS VOLUME ID DAALO
    fsType: ext4
```

PERSISTENT VOLUME

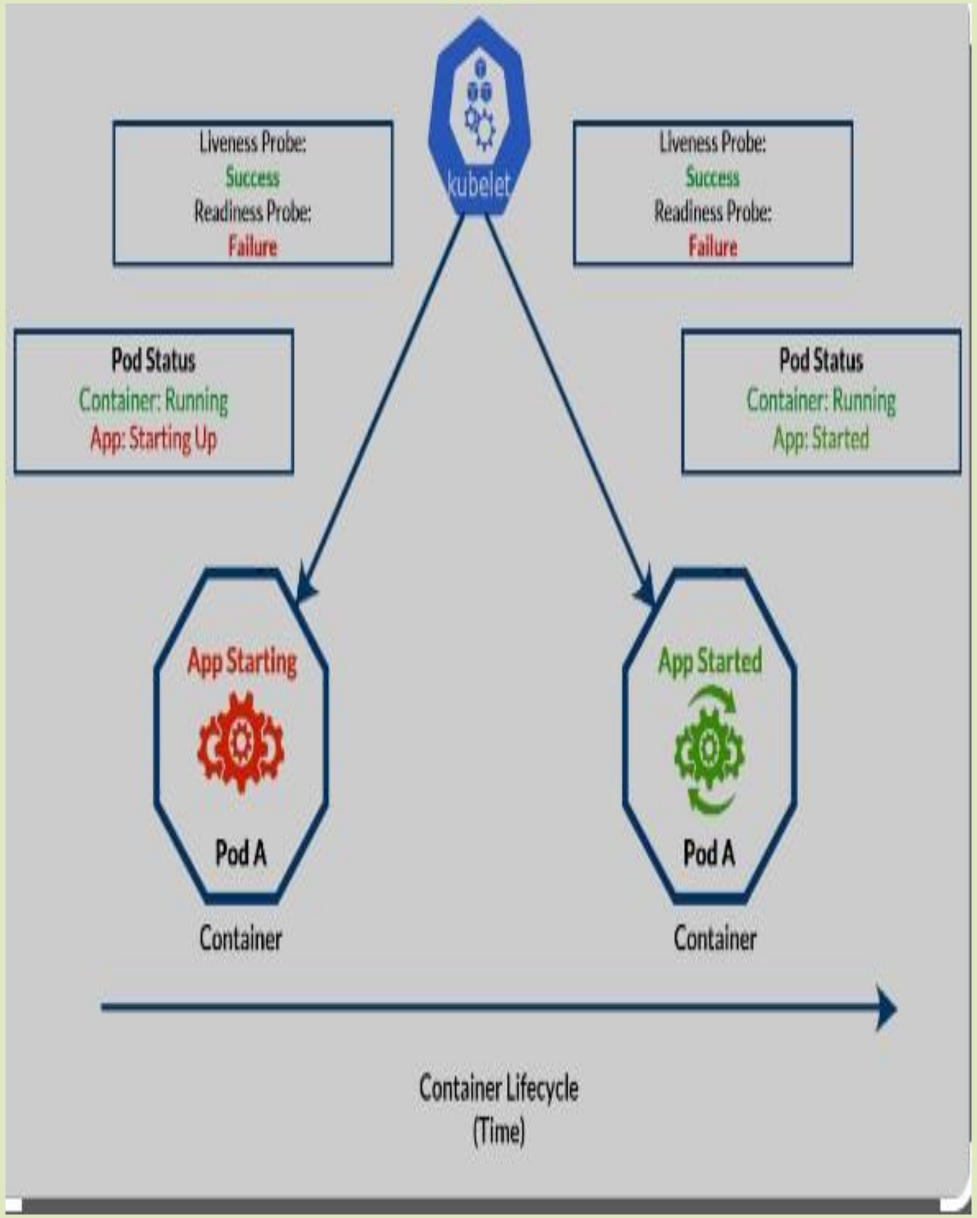
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myebsvolclaim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

PERSISTENT VOLUME

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pvdeploy
spec:
  replicas: 1
  selector:     # tells the controller which pods to watch/belong to
    matchLabels:
     app: mypv
  template:
    metadata:
      labels:
        app: mypv
    spec:
      containers:
      - name: shell
        image: centos
        command: ["bin/bash", "-c", "sleep 10000"]
        volumeMounts:
        - name: mypd
          mountPath: "/tmp/persistent"
      volumes:
        - name: mypd
          persistentVolumeClaim:
            claimName: myebsvolclaim
```

Health check and Liveness Probe

- A Pod is considered ready when all the containers are ready.
- In order to verify readiness of containers in POD to send traffic K8s provides a range of health check mechanism.
- Health checks are carried out by kubelet to determine when to re-create a container for liveness probe and used by Services and Deployment to determine if PODS can send/receive traffic.
- Liveness probe could catch a deadlock when application running but unable to make a progress , restarting that app could help to make application more available.
- One use of Probes(readiness) is to control which PODS are used as backends for services when POD is not ready , its removed from Service Load balancers.
- Commands for Health check return 0,kubelet considers containers alive in this case . For non-zero value , it kills and re-creates it.

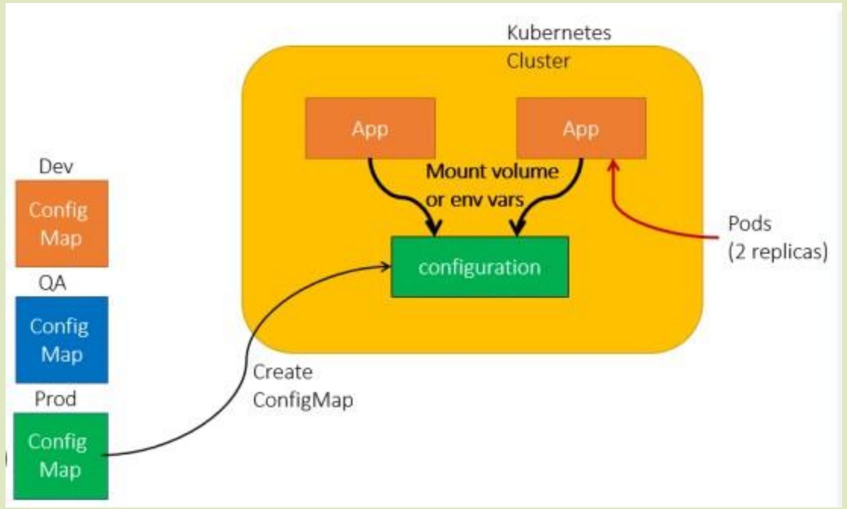


HEALTHCHECK/LIVENESSPROBE

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name:  mylivenessprobe
spec:
  containers:
  - name: liveness
    image: ubuntu
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 1000
    livenessProbe:                                  # define the health check
      exec:
        command:                                    # command to run periodically
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5         # Wait for the specified time before it runs the first probe
      periodSeconds: 5                              # Run the above command every 5 sec
      timeoutSeconds: 30
```

Configmap and Secrets

- While performing app deployments on K8s cluster we may need to check the app configuration file depending on the environment like Dev, QA, Stage or prod.
- Changing the application configuration means we need to change source code, commit change, create a new image and then go through complete deployment process.
- Here, these configuration files should be de-coupled from image content in order to keep containerised application portable.
- This is where K8s configmaps comes handy , it allows us to handle configuration files much more efficiently.
- Configmaps are used to store and share non-sensitive, unencrypted information use secrets otherwise.
- Configmaps can be used to store fine-grained information like individual properties or entire configuration file.
- Configmapfiles are not good replacement for properties file.
- Configmap files can accessed in following ways
- 1→As environment variable
- 2→As Volume in pod
- Kubectl configmap<mapname>- - from-file=<filetoread>abc.cnf



Configmap

```
 apiVersion: v1
kind: Pod
metadata:
  name:  myvolconfig
spec:
  containers:
  - name: c1
    image: centos
    command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]
    volumeMounts:
      - name: testconfigmap
        mountPath: "/tmp/config"   # the config files will be mounted as ReadOnly by default here
  volumes:
  - name: testconfigmap
    configMap:
       name: mymap   # this should match the config map name created in the first step
       items:
       - key: sample.conf
         path: sample.conf
```

Configmap

```
apiVersion: v1
kind: Pod
metadata:
  name: myenvconfig
spec:
  containers:
  - name: c1
    image: centos
    command: ["/bin/bash", "-c", "while true; do echo Technical-Guftgu; sleep 5 ; 
done"]
    env:
    - name: MYENV         # env name in which value of the key is stored
      valueFrom:
        configMapKeyRef:
          name: mymap      # name of the config created
          key: sample.conf
```

Secrets-Lab

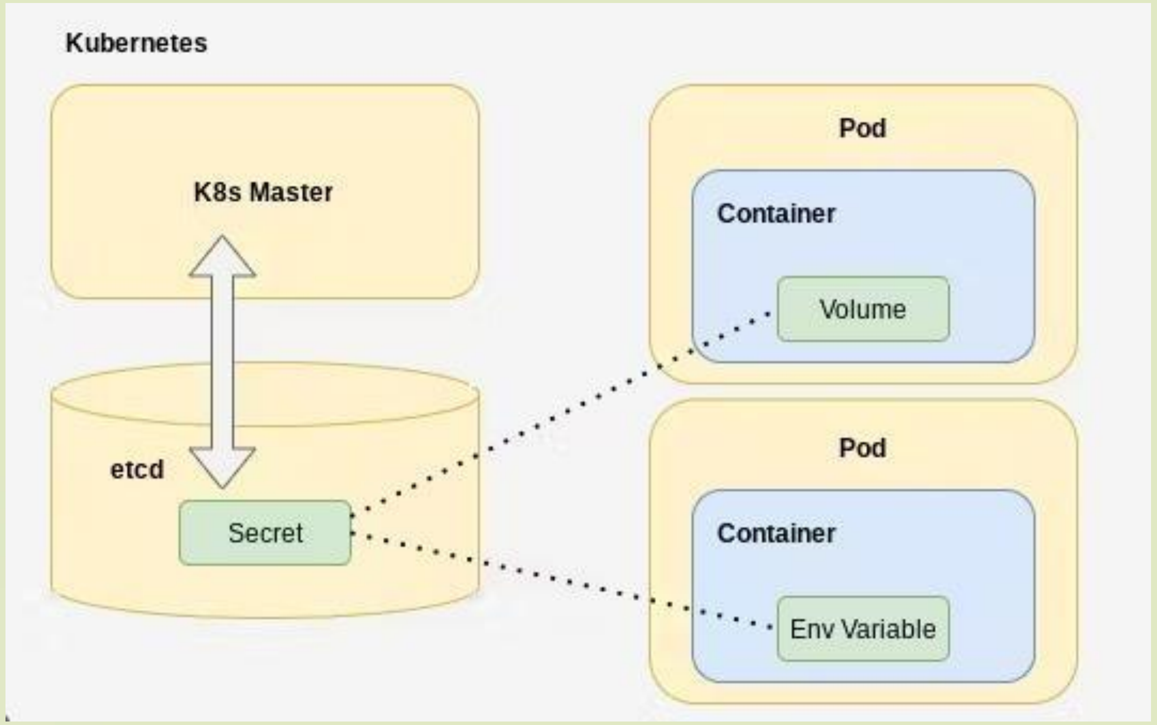
```
echo "root" > username.txt; echo "password" > password.txt

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

```
apiVersion: v1
kind: Pod
metadata:
  name:  myvolsecret
spec:
  containers:
  - name: c1
    image: centos
    command: ["/bin/bash", "-c", "while true; do echo Hi  Ashok; sleep 5 ; done"]
    volumeMounts:
      - name: testsecret
        mountPath: "/tmp/mysecrets"  # the secret files will be mounted as ReadOnly by default here
  volumes:
  - name: testsecret
    secret:
       secretName:  mysecret
```

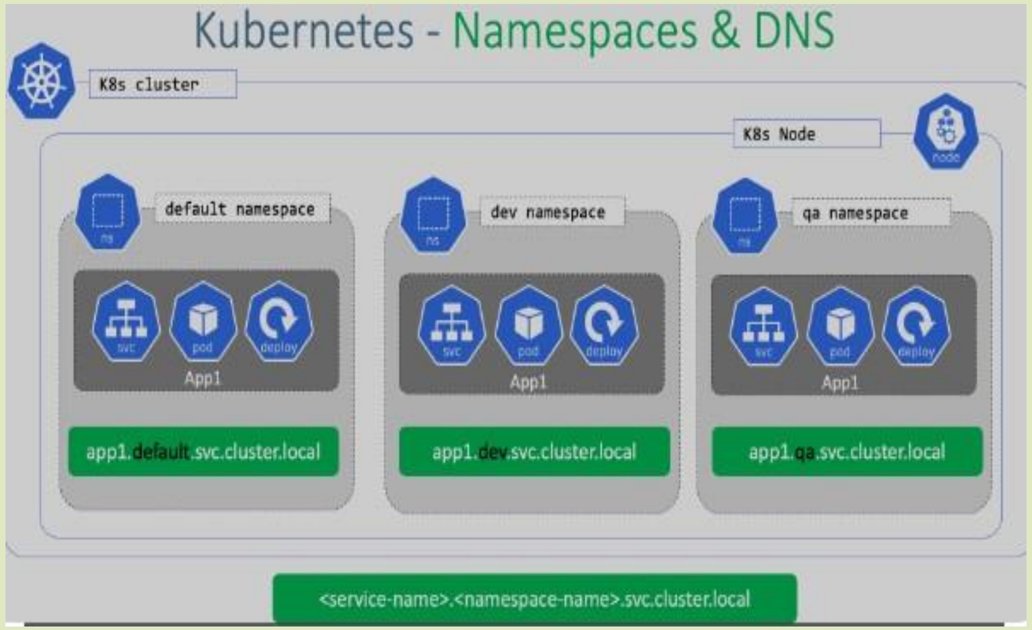
Secrets

- We don’t want sensitive information like database password or an API key kept around in clear text.
- We can access them via volume or environment variable from a container running in pod
- Secret data is stored on nodes in tmpfs volumes(tmpfs)is a file system which keeps file in virtual memory.
- The API server stores secrets as plaintext in etcd.
- Secrets can be created from:-
- 1—from text file
- 2→from a yaml file.



Namespaces, Limits and Requests

- We can name our objects but if many are using same object then it would be difficult to manage.
- A namespace is a group of related elements that have a unique name or identifier.
- Each project has a namespace inside which pods are created , If project B has namespace , project A can't see project B without namespace.
- If no namespace is defined for a cluster then default namespace is given i.e. default namespace.
- If we give command kubectl get pods and no pod is present then it gives output –"No resource found in default namespace"
- It's area or folder of my project where I m working.
- We can define Resource quota for namespace,how much CPU,RAM,Storage is defined for them,so that resource wastage doesn't happen.
- It provides a mechanism to attach Authorization and policy to sub section of the cluster.
- We can use Resource Quota on specifying how much resource each namespace can use.


Kubernetes - Namespaces & DNS
K8s cluster
K8s Node
default namespace
dev namespace
qa namespace
App1
App1
App1
app1.default.svc.cluster.local
app1.dev.svc.cluster.local
app1.qa.svc.cluster.local
<service-name>.<namespace-name>.svc.cluster.local

Namespaces, Limits and Requests

- Most K8s resources(Pods, Services, Replication-controller) are in same namespace and low level resources such as nodes and persistent volume are not in any namespace.
- Namespaces are intended for use in environments with many users spread across multiple teams or projects. For clusters with a few to ten ,number of users, it's not required.
- If we have shared K8s clusters for DEV and Prod use cases , Dev Team would like to maintain space in cluster where they can view a list of PODS, Services, Deployments with no restriction on modification to enable Agile Development.
- For Prod Team, we can enforce strict procedure on who can or cannot manipulate set of PODS,Services and Deployment.
- `$ kubectl config set-context $(kubectl config current-context) --namespace=dev`
- `$ kubectl config view | grep namespace:`

Manage Compute Resources For Containers

- A POD in K8s will run with no limit on CPU and memory
- We can optionally specify how much CPU and Memory(RAM)each container needs
- Scheduler decides about which nodes to place pods , only if node has enough CPU resources available to satisfy POD CPU requests.
- CPU is specified in terms of cores and memory in unit of bytes
- Two types of constraints can be set for each resource type-
- 1→Request
- 2→Limits
- A Request is that amount of a resource which system will guarantee for the container and K8s will use value to decide on which node to place pods
- A Limit is max amount of resources,K8s will allow container to use.
- In the case Request is not set for a container,it defaults to Limits.
- If Limits is not set,then it defaults =0
- CPU Values are specified in ‘millicpu’ and memory in MB

Namespaces Lab

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev
  labels:
    name: dev

==================================to create a pod=================
vi pod.yml


kind: Pod
apiVersion: v1
metadata:
  name: testpod
spec:
  containers:
    - name: c00
      image: ubuntu
      command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]
  restartPolicy: Never
```

Namespaces-Resources Lab

```
apiVersion: v1
kind: Pod
metadata:
  name: resources
spec:
  containers:
  - name: resource
    image: centos
    command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]
    resources:
      requests:
        memory: "64Mi"
        cpu: "100m"
      limits:
        memory: "128Mi"
        cpu: "200m"
```

Resource Quota

- A K8s cluster can be divided into namespaces,if a container is created in a namespace that has a default CPU,Container does not specify its own CPU Limit, then container is assigned default CPU limit.
- Namespaces can be assigned Resource quota objects,this limits amount of usage allowed to objects in that namespace.
- 1→Compute
- 2→Memory(RAM)
- 3→Storage
- Here are two restrictions that a resource quota imposes on a Namespace
- 1→Every Container that runs in the name space must have its own CPU Limit.
- 2→The total amount of CPU used by all containers in the namespace must not exceed a specified limit.

RESOURCEQUOTA

```
apiVersion: v1
 kind: ResourceQuota
 metadata:
     name: myquota
spec:
    hard:
      limits.cpu: "400m"
      limits.memory: "400Mi"
      requests.cpu: "200m"
      requests.memory: "200Mi"
```

RESOURCEQUOTA

```
apiVersion: v1
kind: LimitRange
metadata:
  name:  cpu-limit-range
spec:
  limits:
  - default:
      cpu: 1
    defaultRequest:
      cpu: 0.5
    type:  Container
```

RESOURCEQUOTA

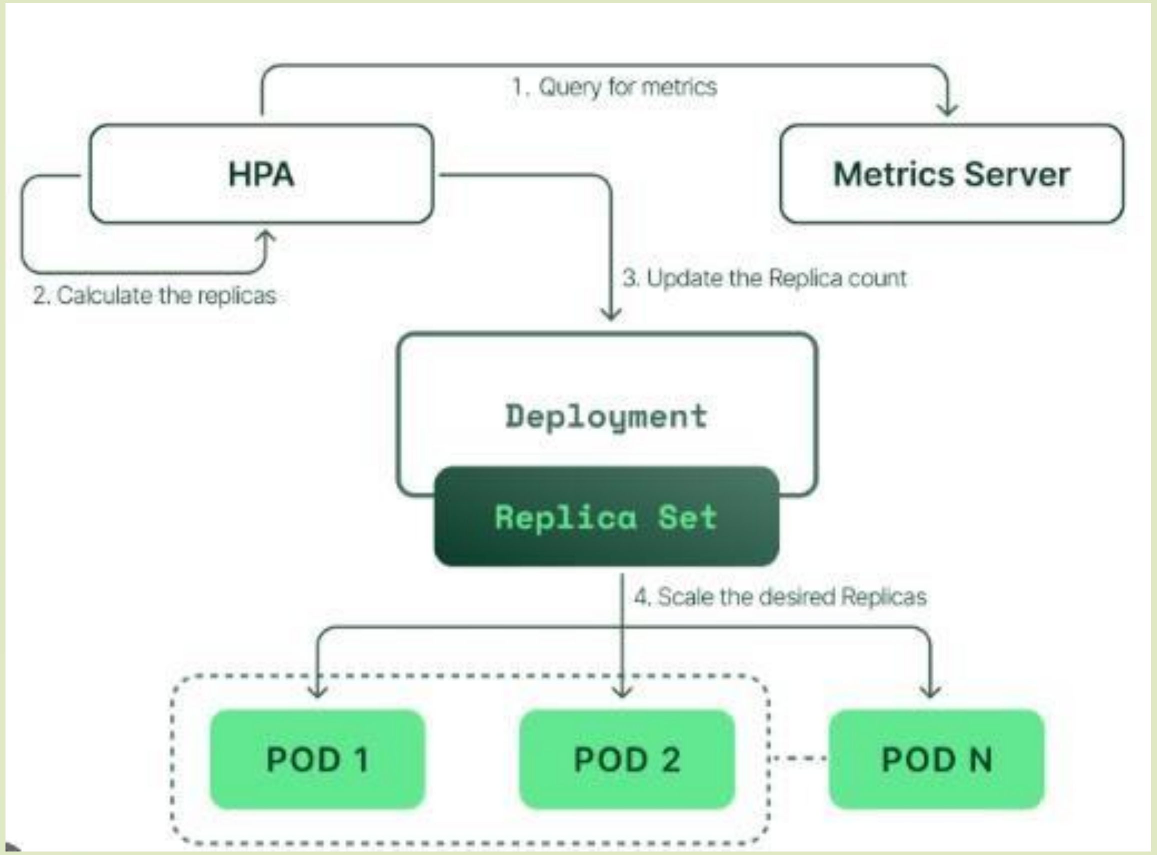
```
apiVersion: v1
kind: Pod
metadata:
  name:   default-cpu-demo-2
spec:
  containers:
  -  name:  default-cpu-demo-2-ctr
    image:  nginx
    resources:
      limits:
        cpu: "1"
```

RESOURCEQUOTA

```
apiVersion: v1
kind: Pod
metadata:
  name:   default-cpu-demo-3
spec:
  containers:
  -  name:  default-cpu-demo-3-ctr
    image:  nginx
    resources:
      requests:
        cpu: "0.75"
```

Horizontal POD Scaling

- Kubernetes has the capability to automatically scale PODS based on observed CPU Utilization which is horizontal POD scaling.
- Scaling can be done only for scalable objects like controller , Replicaset or deployment.
- HPA is implemented as a K8S API resources and a controller.
- The Controller periodically adjust the number of replicas in a replication controller or deployment to match the observed average CPU utilisation to target specified by user


1. Query for metrics
HPA
Metrics Server
2. Calculate the replicas
3. Update the Replica count
Deployment
Replica Set
4. Scale the desired Replicas
POD 1
POD 2
POD N

Horizontal POD Scaling

- For per POD resource metrics(like CPU),the controller fetches the metrics from resource metrics API for each POD targeted by HPA.
- Then if target utilization value is specified by user , the controller calculates the utilization value as % of equivalent resource request on each containerin each POD
- $ kubectl autoscale deployment mydeploy --cpu-percent=20 --min=1 --max=10
- Cool down period to wait before another downscale operation can be performed is controlled by - -horizontal-pod-autosclaer-downtime-stabilisation flag
- Default value 5 minutes

Jobs ,Init& Pod Life Cycle

- Jobs are object for specific purpose.
- We have replicasets ,Deployment ,Daemonsets ,Statefullsets ,they all share one common property they ensure ,their pods are always running ,if it fails , the controller restarts ,it recreates it to another node to make sure the application the pods is hosting keeps running.
- If we don't want pod to get recreated once purpose done , JOB object ensures it doesn't get created.

- Use Cases:
- 1→Taking backup of a DB
- 2→Running batch processes.
- 3→Run a task at a scheduled interval
- 4→Log Rotation
- Job does not get deleted on its own , we have to delete it.

Jobs-Lab

```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  template:
    metadata:
      name: testjob
    spec:
      containers:
      - name: counter
        image: centos:7
        command: ["bin/bash", "-c", "echo Hi Ashok; sleep 5"]
      restartPolicy: Never
```

Jobs-Lab

```
apiVersion: batch/v1
kind: Job
metadata:
  name: testjob
spec:
  parallelism: 5                          # Runs for pods in parallel
  activeDeadlineSeconds:  10   # Timesout after 30 sec
  template:
    metadata:
      name: testjob
    spec:
      containers:
      - name: counter
        image: centos:7
        command: ["bin/bash", "-c", "echo Technical-Guftgu; sleep 20"]
      restartPolicy: Never
```

The Cron-Job Pattern

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
 name: Ashok
spec:
 schedule: "* * * * *"
 jobTemplate:
   spec:
     template:
       spec:
         containers:
         - image: ubuntu
           name: Ashok
           command: ["/bin/bash", "-c", "echo Hi Ashok; sleep 5"]
         restartPolicy: Never
```

Init Container

- Initialising Container.
- Init Containers are specialised containers that run before app containers in pod
- Init containers are always run to completion.
- If a pod’s init container fails ,Kubernetes repeatedly restarts the pod until the init container succeeds.
- Init container does not support readiness probe.
- **Use-Cases**
- 1→Seeding a database
- 2→Delaying app launch until dependencies available
- 3→clone a git repository into a volume
- 4→Generate configuration file dynamically

Pod Lifecycle

- Pending→Running→Suceeded→Failed→Completed→Unknown

- **Pending**→
- (A)-The Pod has been accepted by the k8's system but its not running till master stores info in etcd and asks it to allocate node and download image,e.g ubuntu
- (B)-One or more image is still downloading.
- (C)-If Pod cannot be scheduled because of resource constraints.
- **Running**→
- (A)-The Pod has been bound to node.
- (B)-All of the containers have been created.
- (C)-At least one container is still running or is in the process of starting or restarting
- **Succeeded→**
- (A)-All Containers have terminated in success and will not be started.

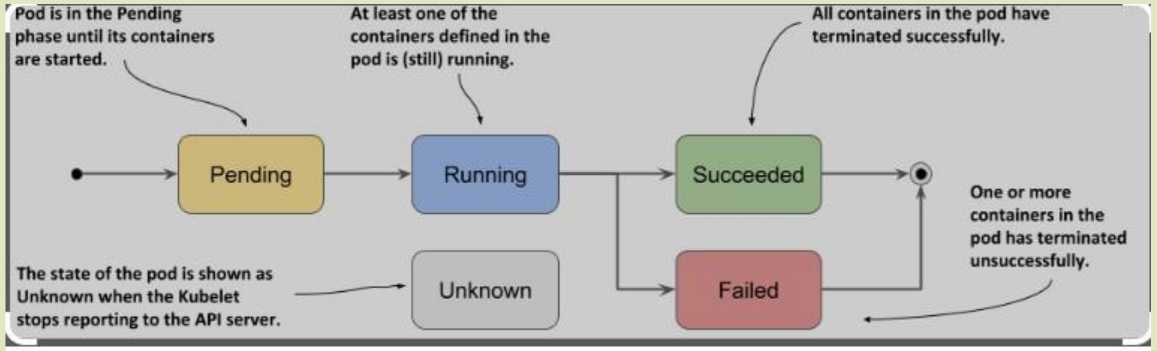
Pod Lifecycle

- **Failed→** All containers have terminated or atleast one container has terminated in failure.

(B)-The container either exited with non-zero status or was terminated by system.

Unknown→(A)State of POD cannot be obtained possibly due to error in Network or communicating host in the pod.

Completed→The Pod has run to completion as there is nothing to keep it running e.g. Completed jobs.



Pod is in the Pending phase until its containers are started.
At least one of the containers defined in the pod is (still) running.
All containers in the pod have terminated successfully.
Pending
Running
Succeeded
One or more containers in the pod has terminated unsuccessfully.
The state of the pod is shown as Unknown when the Kubelet stops reporting to the API server.
Unknown
Failed

POD-Conditions

- The Pod has Pod Status which is an array of- Pod Conditions through which Pod has passed or has not passed
- Using “kubectl describe pod Podname”,we can get pod conditions.
- These are possible types:-
- PodScheduled:- The Pod has been scheduled to a node.
- Ready:-The Pod is able to serverequest and will be added to load balancing pools of all matching Services.
- Initialized:- All Init Containers have started successfully

- Unscheduled:-The Scheduler cannot schedule the Pod right now e.g. due to lack of resources or other constraints.
- Container-Ready:-All Containers in POD are ready