Kubernetes

Kubernetes is Container Orchestration platform.

Containers are Ephemeral in nature, i.e short living. Can die and revive easily.

For example if the number of containers are more on docker platform, It may cause shortage of the resources for last container, when it is not started, it might not got allotted the memory and space even.

One container may die due to another container indirectly because of the excess use of the memory and resources.

**Problem 1:** The nature of the host is scoped to single Host only.

**Problem 2:** Whenever the container is stopped running. It can not be brought up automatically by docker platform.

Restarting the container automatically is known as auto healing.

**Problem 3:** Auto scaling.

When there is more load on the container. Other containers need to be created manually or automatically. To create/delete containers automatically based on the load is called Auto-scaling. Docker doesn’t do the Auto-scaling.

**Problem 4:** Enterprise level support

Docker doesn’t provide enterprise level support. Like

* + Autohealing
  + Auto scaling
  + Firewall
  + Load balancer etc

Q) **What is the difference between the Docker and Kubernetes?**

**Solution for Single host problem**:

Kubernetes creates clusters. It follows master-worker architecture. When a faulty container trouble another container in one node then Kubernetes allocates space in other node. With this single host issue is resolved in Kubernetes.

**Solution to Auto-scaling:**

Kubernetes uses replication controller, replica sets to increase the containers/pods whenever the demand is more.

**Solution to Auto-healing:**

Whenever the container goes down immediately Kubernetes roll out new container. Kubelet will do it with the help of Control plane components.

Docker is never used in production independently. It can be used in production with Docker swarm. It’s not a enterprise solution.

**Enterprise level support:**

Kubernetes is used for enterprise level support. It is widely used across the organizations.

- By default Kubernetes doesn’t support advanced load balancing. It follows round robin loading balancing method.

- Kubernetes is advancing day by day with good community contributions.

Docker supports Dockershim container runtime only.

Whereas Kubernetes supports containerd, dockershim, crio etc as container runtimes.

**Worker Node components/ Dataplane**

**Kubelet** is responsible for creation of the pod and checks the pod is running or not. If the pod goes down it restarts the pod using the help of Kubernetes control plane.

**Kube-proxy:** takes care of networking of the pods like generating the IP addresses and load balancing, when there are more replicas of the pod, it load balances the traffic among the pods.

Worker Node consists of:

* Container runtime
* Kubelet
* Kube-proxy

Basically using above components we can run the application.

**Master Node components/ Control Plane**

**API Server:** It is heart of the Kubernetes, it exposes the Kubernetes to the external work and it receives the requests.

**Scheduler:** It is responsible to schedule the pods and Kubernetes resources on the nodes. Based on the requests received to the API Server.

**etcd:** is the key value store. Entire Kubernetes cluster information is stored in it.

**Controller Manager:** In Kubernetes there are multiple controllers like replica sets, Control manager controls them.

**Cloud Control Manager:** For each cloud provider, to allow the Kubernetes to run on their cloud, certain code logic need to be implemented on CCM. CCM is open-source code, any cloud provider can add their use it for using Kubernetes on their cloud.

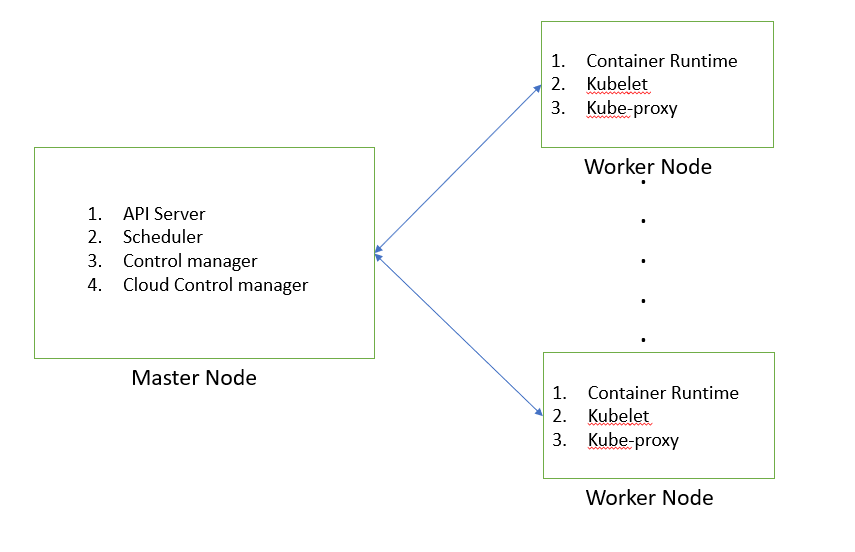


Figure 1 : Kubernetes Architecture diagram

**Kubernetes distributions:**

Minikube, k3s etc are Kubernetes in development environment, these are not full blown distributions.

Kubernetes is the opensource, The distributions of it are developed on the top of it. Like

EKS, AKS, Openshift, Tanzu, Ranchet etc. These are supported by various organizations. They provide customer support for these.

The widely used distribution of Kubernetes is Kubernetes itself. It’s not supported by any organization. So it is used by many organizations.

To avoid more cost to project, It is used in staging and pre-prod environments. For productions they can go with other distributions.

Widely used Kubernetes distributions are:

1. Kubernetes
2. Openshift
3. Ranchet
4. Tanzu
5. EKS
6. AKS
7. GKE
8. DKE

To manage the hundreds of Kubernetes clusters, KOPS can be used , It’s nothing but Kubernetes operations.

Earlier kubeadm is used instead of KOPS, but it has lot of manual operations to do. Now a days KOPS is used widely.

To manage the life cycle of Kubernetes KOPS is used. Like Creation, Upgradation, Managing and deletion of the clusters.

**To create the cluster:**

kops create cluster --name=demok8scluster.k8s.local --state=s3://kops-abhi-storage --zones=us-east-1a --node-count=1 --node-size=t2.micro --master-size=t2.micro --master-volume-size=8 --node-volume-size=8

**Build the cluster:**

kops update cluster demok8scluster.k8s.local --yes --state=s3://kops-abhi-storage

**Validate the cluster:**

kops validate cluster demok8scluster.k8s.local

For Production environment instead of local it will come the domain name like amazon.com etc.

**What is POD?**

POD can have one or more containers in it.

Pod is nothing but a wrapper of a container to make the life of devops engineer easy.

In Kubernetes everything is written in YAML. For standardizing the process. To run the Pod we write the docker run -d image commands in the YAML file.

The Containers within the Pod can communicate with each other using local host and share the files easily.

Container usually have the Side-car, init containers in the Pod along with the application container.

**Kubectl:**

Kubectl is the command line of the Kubernetes, It’s like docker CLI. We use kubectl commands to get the nodes, services and other details of Kubernetes.

Ex: kubectl get nodes

Minikube, kind, k3s, microk8s are the some Kubernetes services that are used for development purpose.

Kind- Kubernetes in docker : It can create 100s of Kubernetes clusters in docker.

Install kubectl

Install minikube

Minikube helps to create to single node cluster.

Command: minikube start

minikube status

We can set virtualization environment to minikube to use.

Pod.yml consists the commands to create pod. It’s a speicification of docker container.

#To create a pod

kubectl create -f pod.yml

#to get pods

kubectl get pods

kubectl get pods -o wide

kubectl get pods -w #watching state

ssh into master or worker node IP addresses

curl <IP Address>: You can check the application running on the cluster.

kubectl cheatsheet consists all the commands handy.

#delete pod

kubectl delete pod <pod\_name>

**deployment :**

deployment is a wrapper that provides autohealing, auto scaling for the pod.

In realtime production environment we don’t deploy the pods, we deploy the deployments, stateful sets, daemon sets.

# To debug the pods

kubectl logs <pod\_name>

kubectl describe pod <podname>

**Kubernetes Deployment:**

The pod is similar to container. It can run multiple containers whereas auto healing and auto scaling can not be performed by pod

Deployments provides the auto healing and auto scaling capabilities. It helps to get zero downtime deployments.

Kubernetes suggest that do not create pod directly. But create it using a deployment resource.

**Deployment creates Replica set(controller), It roll out the pods.**

The replicas mentioned in the deployment yaml manifest are the pods that are available all the time.

Controllers ensures that desired state and actual state of the cluster is same. There are default controller and custom controllers.

**What is the difference between container, pod and deployment?**

What is the difference between deployment and replica set?

Replica set is the controller that provides the auto healing capability. When the deployment is created it’s created automatically to track the replicas in the Kubernetes cluster.

Deployment 🡪 Replica sets 🡪 Pod

#To get all the pods, services, deployments etc

kubectl get all # for a current namespace

kubectl get all -A # for all the namespaces

#Log into the remote Kubernetes cluster

ssh -i <node name/IP>

#to get deployment

kubectl get deploy

#To get replica sets

kubectl get rs

kubernetes deployment examples

**Services:**

Services are created on the top of deployment that acts as load balancers.

Example: When deployment created three pods, three IP addresses will be allotted to them. Users can access them using that IPs, when any of the pod goes down, replica set creates new pod, whereas that new pod will be given a new IP address, User may not be able to access the new pod with the old IP he has. To resolve this issue services will be added on the top of the deployments, it acts as load balancers.

Services does:

* Load balancing
* Service discovery
* Expose app to external world

Service tracks the pods using labels, even if the pod goes down the new pod is created with same label. So new IP address of the pod doesn’t cause any issue to access it.

Services use labels and selectors for service discovery.

Service allows the application to access outside the Kubernetes cluster.

Default services:

* Cluster IP – Application is accessible within the cluster
* Nodeport – accessible within the organization who has the access to node.
* Load balancer – External users can access it.

When the user from outside of the organization wants to access the application in the Kubernetes cluster. Cloud Controller Manager generates the Public IP and sends to Load balancer service. Using it user can access the application.

Using Nodeport service only who has the access to the worker node can only access the application.

Kubeshark – Traffic viewer for the Kubernetes. It shows the how the services working and how the communication is happening between the pods etc.

Pod by default connects with cluster network.

**Nodeport Service:**

Your application will be exposed on the Node IP address.

Port mapping is done to the container port to the node port. Using the node ip and the port application can be accessed.

In Service.yml file:

Kind : service

Selector: <label of the pod>

-Cloud control manager generates the external IP address for the container to access it from outside.

Kubectl edit svc <svc\_name>

**What is Ingress in Kubernetes?**

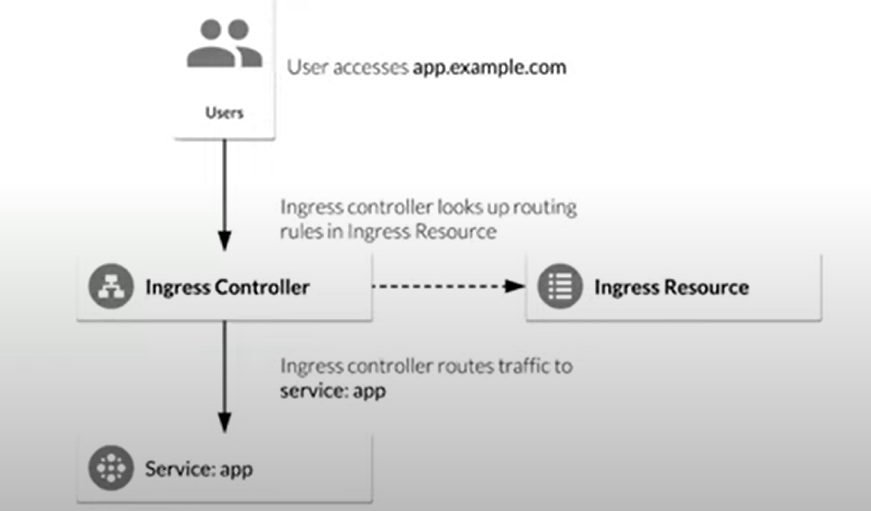
Ingress is a workload API object in Kubernetes which provides external access to the services within a cluster.

It acts as Layer 7 (application) load balancer, allows external traffic to be routed to different services based on http/https routes and rules.

What is the difference between Layer 4 and Layer 7 load balancers?

Layer 4 load balancers routes the network traffic without modifying it or applying any logic to it.

Layer 7 load balancer or application load balancer routes the traffic based on the url or path. It redirects the traffic to different service based on the rule.



In Ingress, Ingress Controller, Ingress Resources are two different objects.

Ingress controller is the heart of the Ingress, which receives the requests from the users. Azure, AWS, GKE or third parties have the Ingress controllers

Ingress Resources has the all the rules where the services mapped with urls/path are defined. Ingress controller takes the decision to route the traffic based on the Ingress Resource.

**DaemonSets:**

**Q) What does Kubelet do?**

It takes the commands from the master node and send them to the Worker node, it ensure those operations are done on Worker node.

Adding the new worker nodes to the cluster is kubelet responsibility

It monitors the health of the Pods in the worker nodes. If any issue happens it informs the master node so that It can bring up the new node parallelly.

Kubelet starts the scheduled pods in the worker nodes. When pod is not functional or not performing efficiently, kubelet will tell the control panel.

**Q)How is Node management is done in Kubernetes?**

Node management work can be done without interrupting the running services in the cluster.

Evict the PODs and safely performs the node maintenance.

To view the Kubernetes Node status:

kubectl get nodes

To Check the PODs status

kubectl get pods -o wide

Following command can safely take the node out from the cluster and evicts the PODs from the Node.

kubectl drain [NODE\_HOSTNAME] -ignore-daemonsets

Above command will transfer the PODs in the node to other nodes.

The node will be disabled for the scheduling. Now The Node can be done with maintenance services.

Once the maintenance operation is completed we can uncordon the Node. Scheduling will be enabled.

\* During deletion of the Node as well all the PODs need to be evicted.

How is POD Execution is done in Kubernetes?

**Storage in Kubernetes:**

In Kubernetes to store the content of the PODs we can use multiple options, One of them is creating the volumes and mounting them with Node where exactly the POD is created using mountPaths.

**POD Definition YAML file With Volumes:**

apiVersion : v1

kind: Pod

metadata:

name: test-pod

spec:

restartPolicy: Never

volumes:

-name: pod-vol

hostPath: var/k8s

containers:

-name : mybusybox

Image: busybox

command: [‘sh’, ‘-c’, ‘echo learning k8s with siva > /siva/k8s.txt sleep 3600’]

volumeMounts:

-name : pod-vol

mountPath: /siva # this path is the path which is created in the POD

In default set up PODs store the data with in it and when the POD dies the data is lost and administrator can’t access it. To avoid this issue storage can be provisioned for it.

PVs acts as storage option for various PODs that are running inside the Kubernetes cluster.

PVs can be created using YAML files.

PV is the piece of storage that is either local or cloud storage.

First PVC connects with PV and then the POD connects with PVC. The binding happens in this manner.

YAML file definition:

kind: PersistentVolume  
apiVersion: v1  
metadata:  
 name: pv-vol-test  
spec:  
 accessModes: [ "ReadWriteOnce" ]  
 capacity:  
 storage: 1Gi  
 hostPath:  
 path: /tmp/local/data

In Production environment: You should not use hostPath

To create PV using yaml file

kubectl create -f pv.yaml

To create list the PVs

kubectl get pv

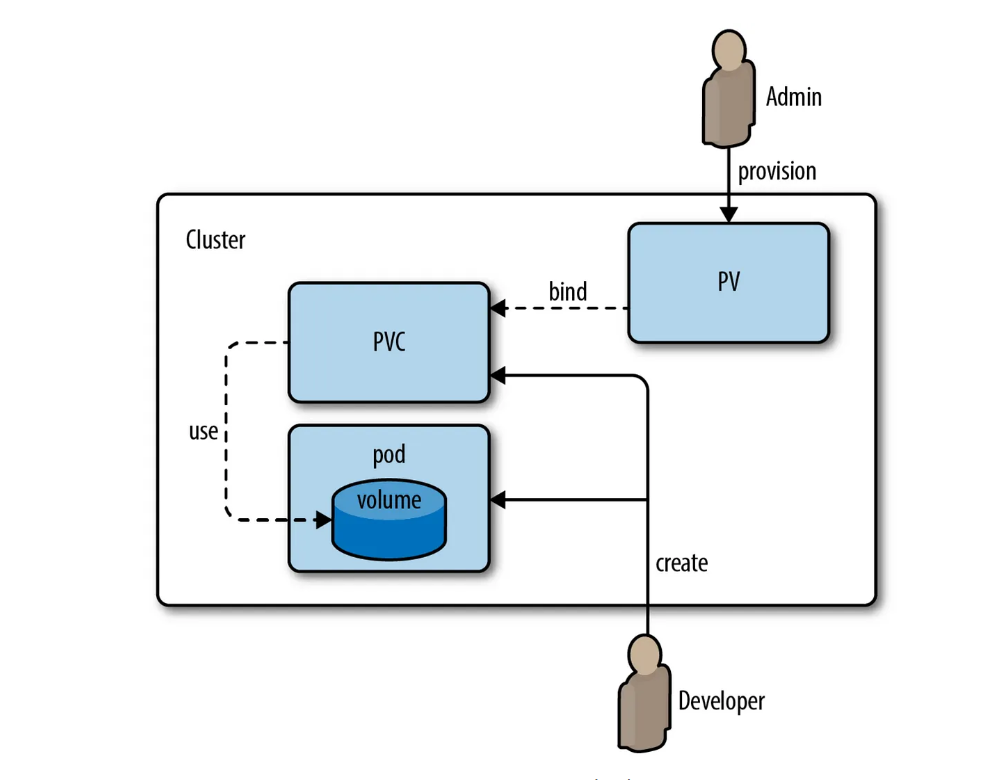
To create PV using yaml file

Kubectl describe pv pv-vol-test

**What is Persistent Volume Claim in Kubernetes (PVC)?**

* PVC is the storage request that submitted by POD
* It asks for storage with certain characteristics. (size, access mode, storage class)
* Kubernetes then matches the request to an available Persistent Volume (PV) and binds them.

POD -> PVC -> PV



In definition file:

Kind : PersistentVolumeClaim

Commands:

kubectl get pvc

kubectl describe pvc test-pvc

POD Definition file to glue the PV to the POD using PVC

apiVersion: v1  
kind: Pod  
metadata:  
 name: mypod  
spec:  
 containers:  
 - name: myfrontend  
 image: nginx  
 volumeMounts:  
 - mountPath: "/var/www/html"  
 name: web  
 volumes:  
 - name: web  
 persistentVolumeClaim:  
 claimName: myclaim

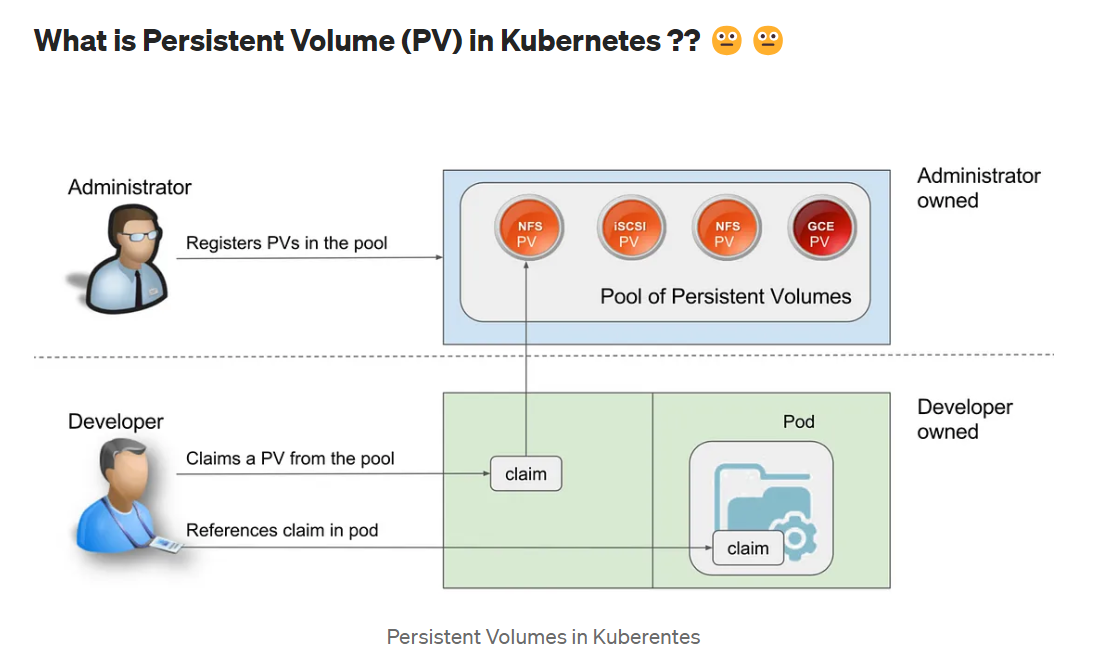
The PVC should be present the in the same namespace as POD, where as PV can be in the cluster.

PVC is name-space scoped

PV is cluster scoped.

**What is Persistent Volume?**

Persistent Volume is pool of storage volumes that are configured by administrators and used by the users deploying the application on the cluster.



**What is Static Provisioning and Dynamic Provisioning of PVs?**

Static provisioning the PVs are provisioned by the administrator manually. Where as in Dynamic provisioning the cluster provisions the PV and allocate it for the POD.

**Storage Class:**

Kubernetes Storage classes is the mechanism that allows dynamically provision of the Persistent Volumes in Kubernetes cluster.

Kubernetes admin specifies class of storage so that the pods can dynamically request specific type of storage they need.

1. Storage class allows dynamic provisioning of Persistent Volumes when PVC claims it.
2. Storage classes abstracts the underlying storage provider.
3. Storage classes used in conjunction with PVC that allow POD to dynamically request new storage.
4. Storage classes use provisioners that are specific storage platform or cloud provider to give the Kubernetes access to the physical storage.
5. Each storage backend has own provisioner. Storage backend is defined in the storage class component via provisioner attribute.

# Command to get storage classes

kubectl get sc

Storage class define the properties of the storage system:

* Speed (SSD, HDD)
* Quality of service levels
* Back up or replication policies
* Type of file system
* Or any property defined by the administrator

Once the Storage class object’s parameters are set, they cannot be updated.

**What are StatefulSets?**

A StatefulSets in Kubernetes is a special controller that manages pods which:

* Need a unique name (like pod-0, pod-1) any time it restarts even.
* Need persistent storage that sticks around.
* Need to start or stop in specific order
  + It means starts pod-0 first after its ready starts pod-1, deletes pod-1 first and later pod-0.
* Should not get randomly replaced or renamed

In pods : mongo-1, monog-2, mongo-3 : mongo-1 is primary it need to be started first before the second one replicates from it.

It’s used in Databases and Messaging queues.

**Analogy**: StatefulSets are like a named locker for the student in the school, it keeps their stuff (data) no matter what.

**What is the difference between Deployment and StatefulSets ?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Deployment** | **StatefulSets** |
| Pod Identity | All pods are same | Each pod has unique, stable name |
| Storage | Shared or None | Each has own persistent volume |
| Order | Pod can start/stop any order | Need to start or stop in specific order |
| Example use | Web, stateless servers | Databases, Kafa..etc |

**What are the Kubernetes Probes?**

Probes are like health checks that Kubernetes uses to check:

1. Liveness Probe -> Is the app still alive or stuck? -> if fails restart the pod
   1. It kills the pod and restarts
2. Readiness Probe -> Is the app ready to receive the traffic - > If fails stop sending the traffic
   1. It removes the pod from the load balancer to stop receiving the traffic
3. Startup Probe -> Has the app finished starting? -> If fails wait for some more time.
   1. **While this probe liveness and readiness are ignored.**

**What are the Rolling Deployments?**

Rolling Deployment are way to update your application gradually, one part at a time. Without downtime.

* Create a new pod with updated version
* Waits until the new pod is healthy
* Delete one of the old pods
* Repeat this one by one until all pods are updated.

**Real-world Analogy:** Think of updating the all the bulbs in the room. You don’t switch off all the lights at once. Replace one by one. So that the room never fully dark.

**What is Blue Green Deployment?**

Blue-Green Deployment is a technique to release the new version of the application without downtime and easy roll back.

We run two identical environments

Blue = Current version live (production)

Green = The new version (staging/test)

When new version is tested and ready you switch the traffic from Blue to Green.

**If the python app is running on a node where redis app should not run on the same node?**

Node Affinity: Python app runs on the labeled node.

Pod AntiAffinity: Teslls redis not to run on the node where app=python.

**What is Node Affinity?**

Node affinity means telling the Pods where to run based on the labels on the nodes.

Nodes are labeled with key value pairs.

Process:

1. First label a node:

kubectl label nodes node1 zone=us-east

1. Use the Node affinity in the Pod.

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: zone

operator: In

values:

- us-east

**What is Pod Affinity?**

Placing the related Pods together. Each pod has labels, we can write the rules based on the labels

Uses:

* Pods communicates frequently, reduce the network delay.
* Microservices working closely
* Caching layer + Service layer : Faster access.

requiredDuringSchedulingIgnoredDuringExecution : Must match (strict rule)

preferredDuringSchedulingIgnoredDuringExecution : Try match if possible (soft rule)

**Explain the etcd Quorom Mechanism? What happens if majority of etcd nodes failed?**

Quorom is majaority of the etcd nodes (More than half)

If number of nodes are N then the quorum is Floor(N/2)+1

If Quorum not met:

* Etcd can not elect a leader
* Kubernetes control plane fails to create create pod, update service or store change state.

Read may work with local data,

Write, Leader election, Cluster healing fails.

Best Practices to avoid Quorum loss:

* Use an Odd number of etcd nodes(3,5,.. etc)
* Never run etcd on unreliable VMs or networks
* Regular backup of etcd nodes
* Distributes etcd nodes across the zones or racks.

**What is Multi-Tenancy in Kubernetes?**

Multi-tenancy is running workloads for multiple users or groups on a shared cluster, while isolating them from each other.

**Namespace:**

* Each tenant gets their own namespace.
* Namespace isolates pods, services, configMaps, secrets, Network policies
* Namespace alone not a strong security boundary. It’s a base layer.

**Role Base Access Control (RBAC):**

* Control who can do what in each namespace
* Prevent access to other tenant’s resources
* In yaml file kind : RoleBinding

**Network Policies:**

* Kubernetes Network policies restrict the pod-to-pod communication between namespaces.
* Use “deny all by default” policy + allow specific access.

**Zero Downtime Deployment Strategies**

* Rolling Updates:
* Blue Green Deployment
* Canary Deployment
  + Release the new version to the small percentage of users (ex : 5%), monitor the metrics and gradually increase the traffic to the new version.