

BRI

## (Kubernetes)

### intro to k8s:

- what is kubernetes
- k8s Architecture
- main k8s Components
- minikube and kubectl - Local setup
- main kubectl commands - k8s CLI
- k8s YAML configuration file.

### Advanced Concepts :

- k8s Namespace - organize your components
- k8s Ingress
- Helm - package manager
- volumes - persisting Data in k8s
- k8s StatefulSet - deploying stateful APPS .
- k8s Services .

### (\* What is Kubernetes :-

- It is a open source Container Orchestration tool.
- It is developed by Google .
- It helps you to manage Containerized application .
- It is used to manage different application , in a thousands of container .
- This may be a VM , on-premises or in cloud or hybrid machine .

\* what problems does k8s solve ?

Ans:-

(\*) The need for a Container Orchestration tool?

- It is used for containing microservices app's instead of containing monolithic app's.
- It is used to increased use of -  
monolithic containers .

(\*) what features do orchestration tools offer:-

Ans:-

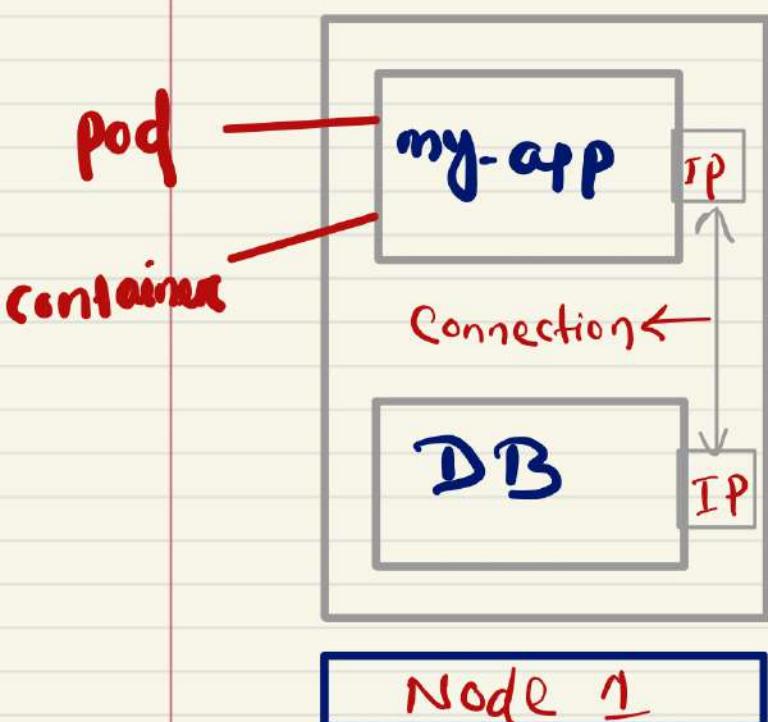
- It provides high availability or no-downtime .
- Scalability or high performance .
- In an disaster recover situation -  
we can take backup and restore .

## Kubernetes Components :-

→ There are different types of Kubernetes Components.

- (1) volumes
- (2) Pod
- (3) Services
- (4) Ingress
- (5) Secrets
- (6) Statefulset
- (7) configmap
- (8) Statefulset
- (9) Deployment
- (10) nodes
- (11) clusters

(\*) pod :-



→ Pod is the smallest unit of the K8S.

→ Usually in one pod we run only one application.

→ Each pod gets its own IP address (internal one).

→ They we tie IP address to communicate.

## ConfigMap :-

External configuration of your appi.

### Secrets :-

- It used to store secret data.
- Base 64 encoded.

### Volume :-

- If in your k8s cluster Pod get restarted then, it's database get also restarted and it's all data goes down.
- So, we use a external storage.
- The storage may be a local machine or remote storage or a cloud storage.

**Ques:-** Do you face any problem/downtime in your application inside in your k8s cluster.

Ans:- Yes, I do have faced multiple problem on down time, in my application inside in my k8s cluster.

- Some time some of my application running in my cluster have goes down time or stopped.
- So for overcome this situation, I used replication setting to replicate my Pod in a different one.

- gt means, g do replicate my all pods which contains my API application and run gt in a separate node.
- Both node's pod connected with same service.
- Services are Two functionalities.
  - permanent IP
  - gt also is a load balancer.
- This replica create by "Deployment" techniques.

The database replicas create by "statefused" techniques.
- For creating extra pods inside in another Node, we used another components called "Deployment".
- In "Deployment", we don't need to create separate pod for replicate pods in another Node.

## Deployment :-

- Blueprint for my-app pods.
- You create Deployments.
- Abstraction of pods.

## Note

- So now if your one application inside in your one node goes down, Then our application doesn't go down, because it is running in our replica Pod - in another Node.
- \* How we replicate our Database inside in a Pod ?

Ans:- We can't replicate our "Database" using Deployment process.

- Because it contains state/Data inside it.
- And for that process we use an another K8s component, called "StatefulSet".
- Using "StatefulSet" we can replicate app-Pod like Database.
- Statefulset applications are like :-

### Database

↳ mongo DB

MySQL

Cassandra and etc.

Note :-

→ "Deployment" is for Stateless APPS.

→ "Statefulset" is for Stateful APPS or Databases.

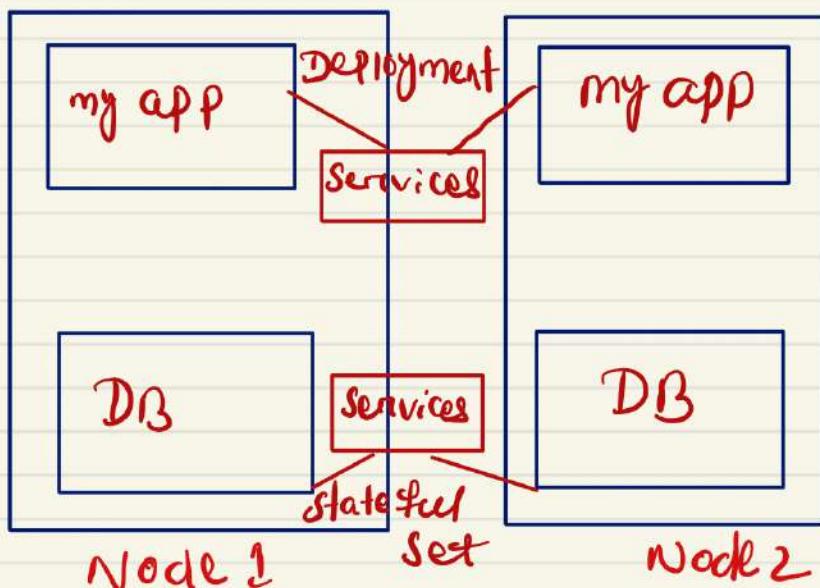
(QMP)

- Hosting Statefulset application is not easy for replication for Databases.
- So it is a best practice that, DB are often hosted outside of k8s Cluster.

(\*)

- So in this way, by using "Deployment" and "Statefulset" techniques in k8s cluster - we can avoid our down time.

→



- So if an application running in my Node 1 - it goes down, then also our app doesn't go down, because it replica running inside in our Node 2/B - Pod.

Main Kubernetes Components :-  
Scattered

Pod :- smallest unit of Node

Abstraction of Container.

Services :-

Establish communication between  
Pods.

Ingress :- Route traffic between clusters.

Configmap :- Create external configuration

Secrets :- Contains secrets.

Volume :- Attach External Volumes or  
Storage in Database.

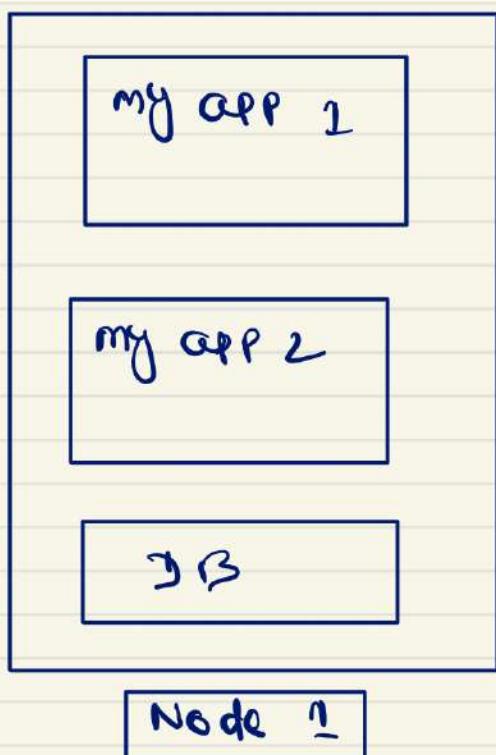
Deployment :- Create replication of  
Pods containing our application.

Statefulset :- Create replication in our  
Database.

## Kubernetes Architecture :-

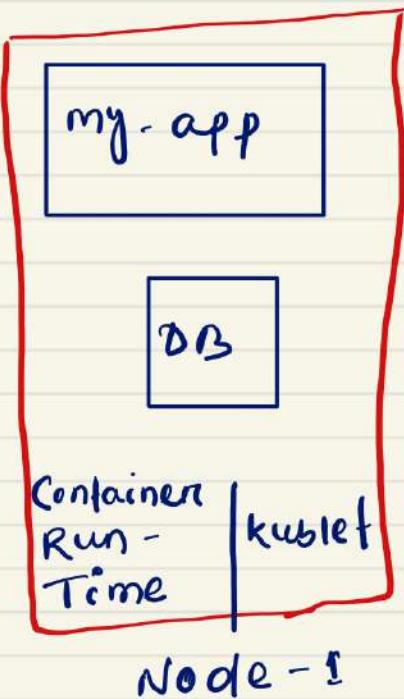
- Kubernetes Architecture mainly based on Node.
- We have two types of node.
  - (a) master node
  - (b) worker node
- In a one node, we have multiple Pod.
- And each Pod contains multiple application.
- Again clusters are again divided in two methodology
  - (1) managed cluster
  - (2) un-managed cluster

## worker machine in k8s cluster :-



- Each node has multiple pods in it.
- 3 process must be installed on every node.
- Worker nodes do the actual work.

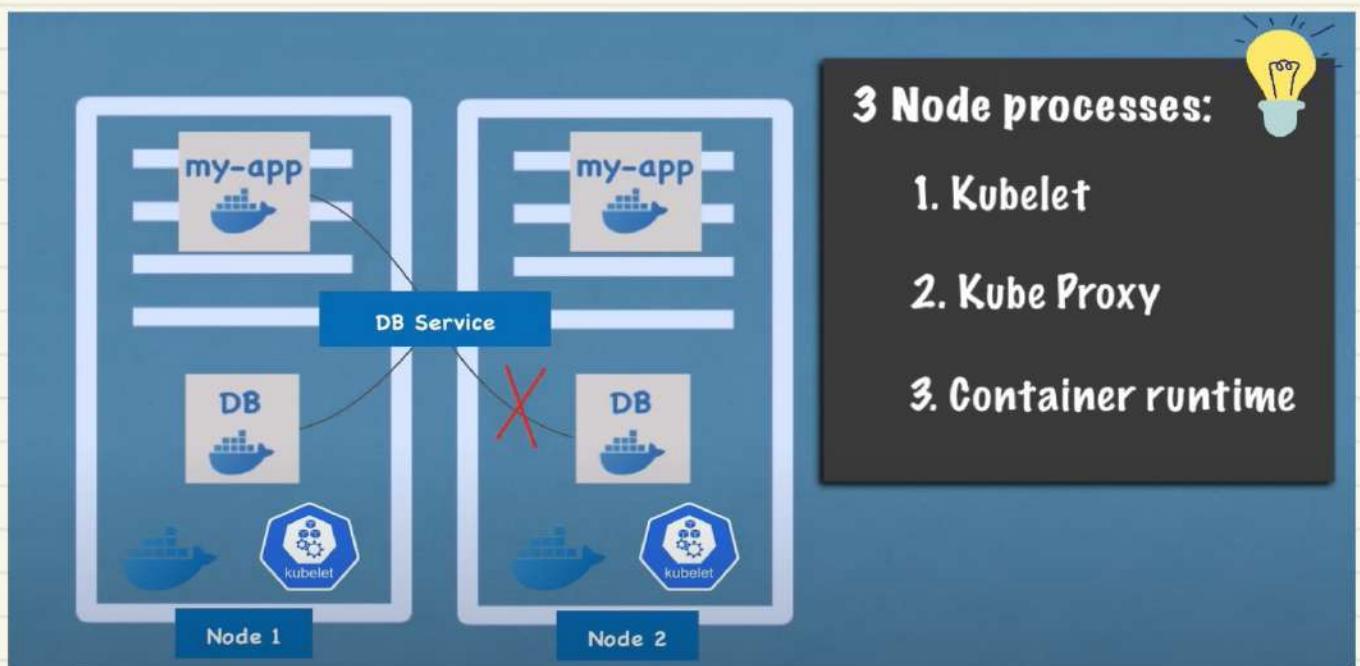
## \* kubelet and Container Runtime :



→ "kubelet" helps to establish a communication between containers and node.

- So in a word, in one k8s cluster, we have multiple "Node" running.
- And inside each "Workers Node" we have again a no. of pod/containers running.
- So "kubelet" helps here to establish connection between pod and node.
- And if we have replica pod running within a another Node, then "Services" is helps to Create communication in between those replica pods.
- So "Services" is the one kind of "Load-Balancer" in here.
- And "kubeproxy" is an another service, who is also helps in Secure Connection establish in between Pods.

→ So "every time", in each "worker node" it is mandatory to install 3 services - which are like :



→ So, if there is any downtime in a cluster, "services" not allow any pod to connect with there another replica pod,

→ It switch the full replica node's pod, to work, in case of any failure or downtime.

gmp: Master Node :-

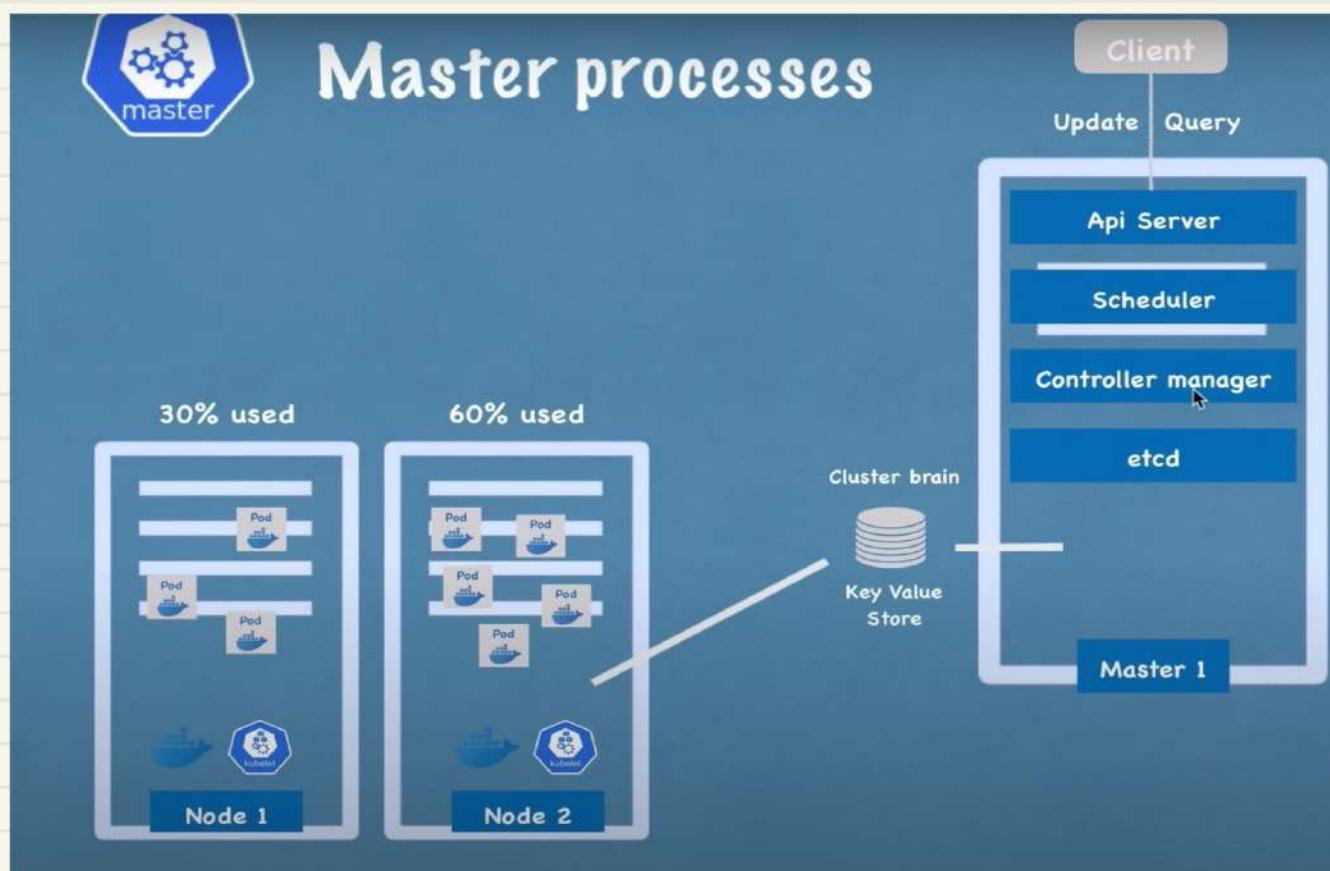
Q:- So, How do you interact with a cluster

- ▷ Schedule a Pod ?
- ▷ monitor ?
- ▷ re-schedule / re-start Pod ?
- ▷ join a new Node ?

Ans:-

So managing process are done by -  
master nodes .

Master Node :-

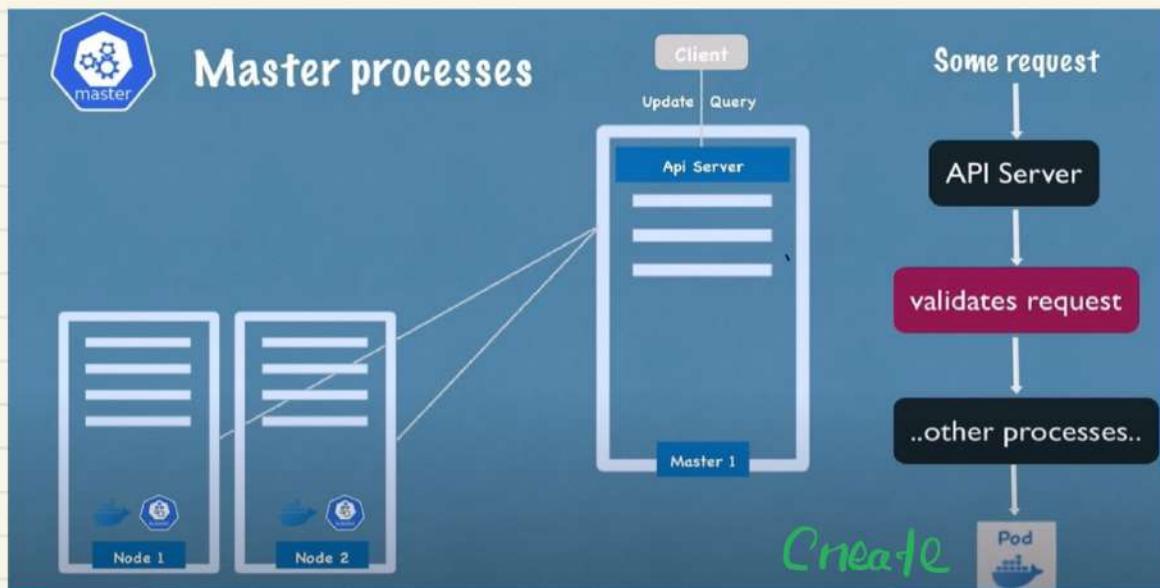


→ In master node mainly we have 4 components

- 1.) API Server
- 2.) Scheduler
- 3.) Controller manager
- 4.) etcd.

### (1) API Server :-

- It is the gateway in cluster.
- It means, if Authenticate, every command before, get installed.
- "API Server" will first authenticate, before creating any pod or new services inside in a node.

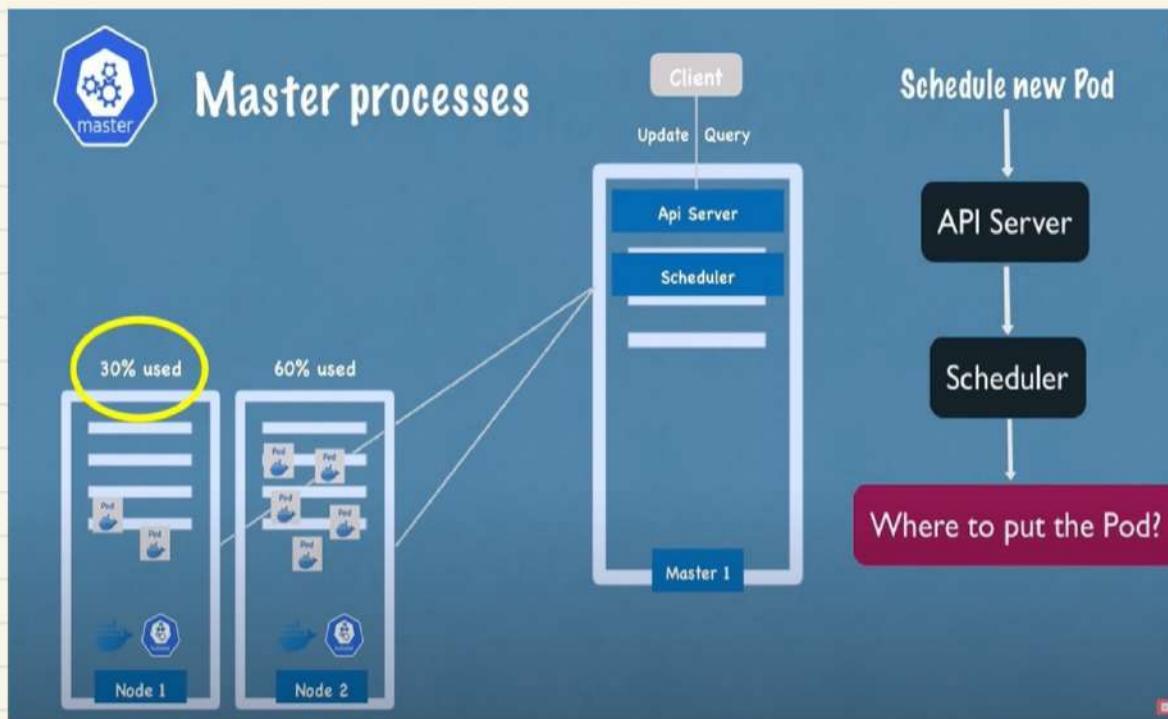


## (g) Scheduler :-

→ Scheduler is the most intelligent tool, who decides, on which pod, need attention to install any services or any new pod, for the current time.

### Example :-

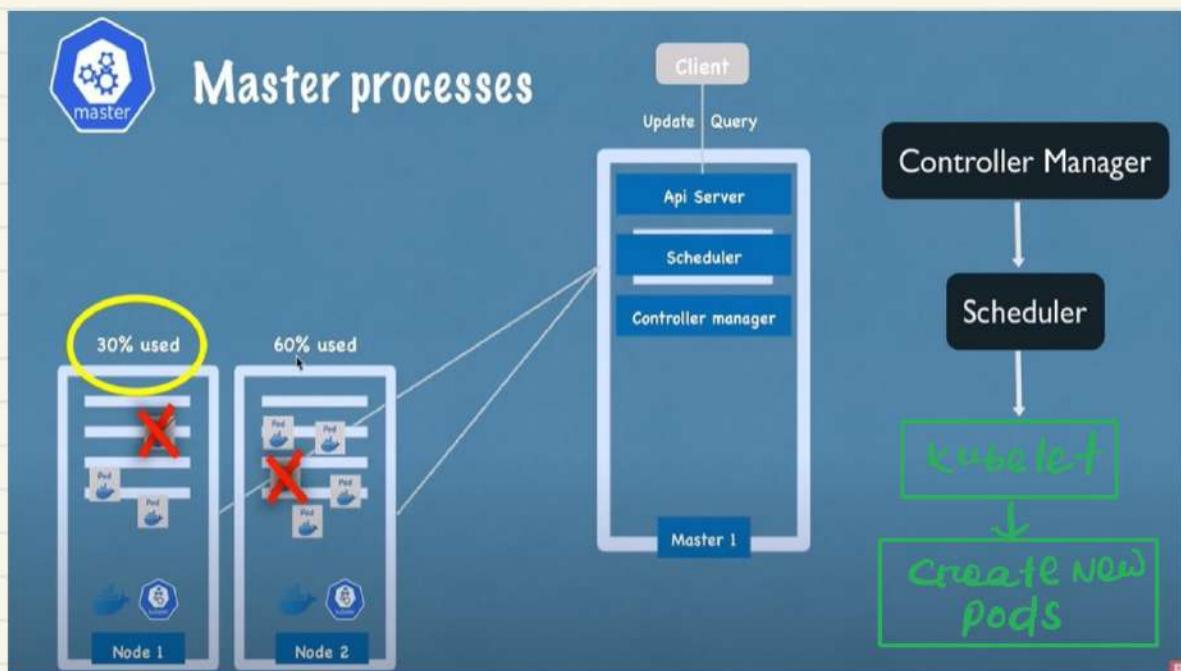
- if you schedule to create a new pod inside a node.
- Then at first it authenticate through API Server, then, it goes to "Scheduler" and scheduler checks, which pod has less work load at the current time.
- Then if he found any pod having less work load, then it schedule that task to that pod.



- So, scheduler will decides that on which node - new pod will be scheduled.

### (3) Controller manager :-

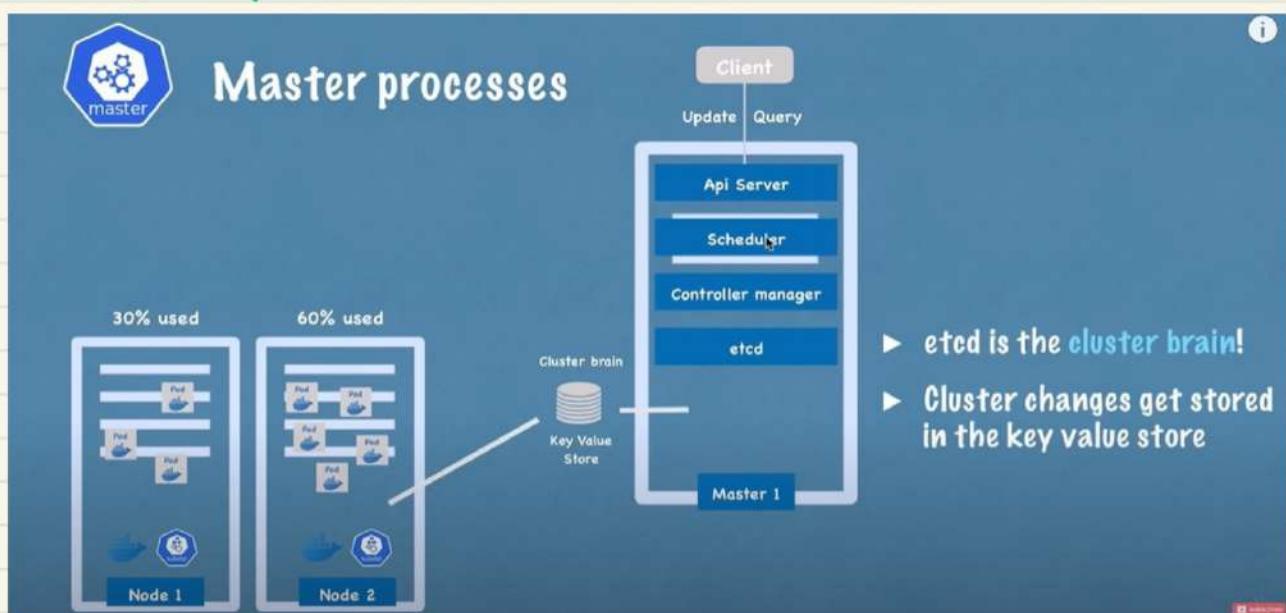
- If any pod gets down or gets stop in any time, then "controller manager" check that stopped pod and start a scheduler.
- And scheduler again checks that stopped pod and it will start "kublet" command to replace or transfer all traffic and connection to an another pod.



- So, finally in one word "controller manager" - detects the stopped "pod" inside in a node.
- And after that, it will restart it's replica pod, by using "Scheduler", and using "kublet" command.

(U) etcd :-

- (i) etcd is the Brain of a cluster.
- (ii) In every node, there is a "key value store" present. who stores all history details - that what all happened inside in a cluster.
- (iii) Inside in a cluster, in "etcd", if have stored, all details like :- when a new pod will schedule to start, when a pod goes down, and when a pod need to restart and many more history will - stored inside in a cluster.
- (iv) So "etcd" is the brain of a cluster.



- (v) So, etcd does n't stored any application or containers data -
- (vi) It is only stored all previous and current work history of a cluster.

## (\*) Example of a cluster set-up :-

- For setup a cluster in k8s, always master-node required less resources like (CPU, RAM, Storage)
- But workers node required more resources as, it handle more application.

## (#) minikube and kubectl setup :-

- ▷ What is minikube?
- ▷ What is kubectl?
- ▷ How to setup minikube cluster?

### ① What is minikube?

Ans:-

- (i) In a production environment, there may be multiple master node and also multiple workers node.
- (ii) So for testing any environment, it requires an environment which has a CPU, memory and etc.
- (iii) So minikube is a extra virtual environment which provides a cluster environment where master and workers node run inside it.
- (iv) Also there is a docker container pre-installed inside it for holding your application.

### Note

- mainly minikube used for, test local cluster setup in-side in a test environment.
- So, minikube creates virtual box environment within in your local setup.
- Then nodes are running in that virtual box.

## Q) what is kubectl?

- kubectl is a command line tool for K8s cluster.
- kubectl is used for creation process of different "Services, Secrets, Configmap" - within inside in a node.
- kubectl always interact with "API Server" first before initiate any command to execute any process.

## \* Some Basic kubectl Commands :-

### CRUD Commands :-

- (1) Create deployment → kubectl create deployment [name]
- (2) Edit deployment → kubectl edit deployment [name]
- (3) Delete deployment → kubectl delete deployment [name]

### Get to know status about different K8s Components

- ① kubectl get nodes
- ② kubectl get pod
- ③ kubectl get services
- ④ kubectl get replicaset
- ⑤ kubectl get deployment

### Debugging. pods :-

- ① Log to Console → kubectl logs [pod name]
- ② Get interactive terminal → kubectl exec -it [pod name] -- /bin/bash

\* For creating Pods in a k8s cluster

Command

kubectl create deployment `image-depl --image`  
`=image`.

Example :-

kubectl create deployment nginx-depl --image=  
`nginx`

\* For generating logs for a Pod in k8s cluster.

Command

kubectl logs pod-name .

\* exec -it

↳ kubectl exec -it pod-name -- bin/bash

(It redirect our Command Line to our Pod )

\* Delete Command

Command

kubectl delete deployment pod-name .

\* .yaml File System :-

→ Every time going one by one Command Line or Providing single single Command value in to a Command , for any Configuration , that is not a best practice .

→ That's Why we use "yaml" file System to Create repeated Configuration each time .

Command

kubectl apply -f config-file.yaml

\* A normal .yaml file for create "nginx" app.

File name := (nginx.yaml)

apiVersion : apps/v1

kind : Deployment

metadata :

    name : nginx-deployment

labels :

    app : nginx

Spec :

    replicas : 1

    Selector :

        matchLabels :

            app : nginx

    Template :

        metadata :

            labels :

                app : nginx

        Spec :

            Containers :

                - name : nginx

                image : nginx:1.16

            Ports :

                - containerPort : 80

\* For executing this (.yaml) file, we use "apply" command.

↳ kubectl apply -f nginx.yaml

Note

If anything required to change in the pod, then only thing required to change the configuration in .yaml file.

## \* Some useful "kubectl" Commands :-

### CRUD Commands

- (a) Create deployment → kubectl create deployment
- (b) Edit deployment → kubectl edit deployment [name]
- (c) Delete deployment → kubectl delete deployment [name]

### Status of different k8s Components

- (d) kubectl get nodes | pod | services | replicaset | deployment

### Debugging pods

- (e) Log to console → kubectl logs [pod name]
- (f) Get Interactive Terminal → kubectl exec -it [Podname] -- bin/bash
- (g) Get info about pod → kubectl describe pod [Podname]

### use configuration file for CRUD

- (h) Apply a configuration file :-  
↳ kubectl apply -f [filename]
- (i) Delete with configuration file :-  
↳ kubectl delete -f [filename]

## YAML Configuration File in Kubernetes :-

- (i) No need to confuse, when you see a YAML file structure.
  - (ii) Basically a (.yaml) file consists of 3-parts of configuration file.
  - (iii) This is like :=  
It connects from "Deployment to Services to pods".

\* 3 parts of a "yaml" file.

→ nginx-deployment.yaml

→ nginx-service.yaml

```
! nginx-deployment.yaml x
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx-deployment
5   labels: ...
6 spec:
7   replicas: 2
8   selector: ...
9   template: ...
10
11
12

! nginx-service.yaml x
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: nginx-service
5 spec:
6   selector: ...
7   ports: ...
8
9
10
11
12
```

→ So first part is  
① metadata

→ so second part is  
② Specification

- Now the third part is "**status**".  
\* And this Status part is auto generated - by the Kubernetes Side.
- In Kubernetes, there is a self-reconciliation process available.
- It means K8s first compare with "Desired and Actual" Status, and if he found any mis-match in states, it will fix the states.

**Desired ?  $\neq$  Actual ?**

**Desired ? = Actual ? (fixed by K8s)**

- (\*) Where does K8s get this status data :-
- Kubernetes has a cluster brain, which is called "**etcd**".
- In "**etcd**" all history of cluster, get stored in the key value store.
- So "etcd" holds the current status of any K8s components.

Format of the ".yaml" configuration files :-

- The extension of the configuration file is **".yaml"**.
- Syntax :- It follows strict indentation.
- So, if you have a very big file, it is best to use a online YAML validator website to check whether every thing is correct or not.

## Layers of Abstraction :-

1. Deployment manages a Replicaset.
2. Replicaset manages a Pod.
3. And a pod is an abstraction of a container.

### \* Labels and Selectors :-

- So "deployment" part has "Labels".
- And "Service" part has "selectors".

### \* Ports in Service and Pod :-

- we have two ports inside in our k8s YAML - configuration file.
- One is the port no. assigned to Pod, in "Deployment" stage.
- And another port no. in "Service" stage.
- And these two port (Container Port and DB port) will connected through a "target port".
- And the target port is same as the - container port.

### Command

How to check endpoint IP and ports in a cluster :-

↳ kubectl describe service [Pod name]

How to check more details of a Pod.

↳ kubectl get pods -o wide.

How k8s generate auto-generated Status file :-

Command :-

kubectl get deployment[Podname] -o yaml

\* How to save it through a file :-

kubectl get deployment[Podname] -o yaml > filename.yaml

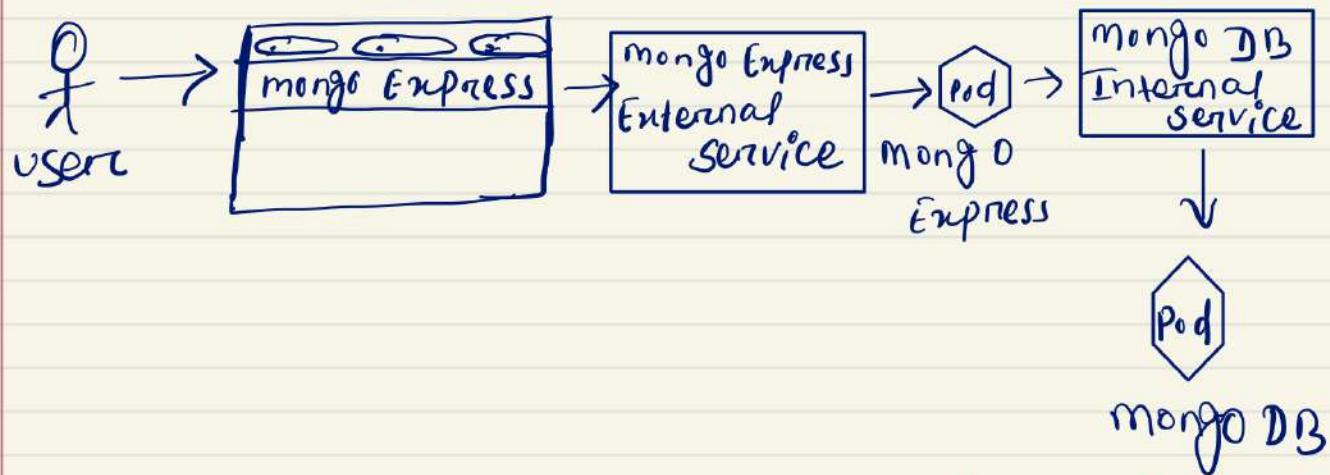
Demo Project :-

Complete Application setup with k8s Components :-

→ Here we will setup two application

(1) mongo-express

(2) mongo DB



Mongo DB application deployment .yaml file :-

mongo.yaml

apiVersion : apps/v1

Kind : Deployment

metadata :

name : mongodb-deployment

Labels :

app : mongodb

Spec :

replicas : 1

Selectors :

matchLabels :

app : mongodb

template :

metadata :

labels :

app: mongodb

spec :

containers :

- name : mongodb

- image : mongo

ports :

- containerPort : 27017

env :

- name : MONGO\_INITDB\_ROOT\_USERNAME

- value :

- name : MONGO\_INITDB\_ROOT\_PASSWORD

- value :

Note :-

- For each time our deployment .yaml file execute it automatically connect with our mongoDB - application.
- So, for successfully login to our mongoDB - application, we need to create a "secret file".

Secret Configuration File :-

`mongo-secret.yaml`

apiVersion : app/v1

kind : secret

metadata :

- name : mongodb-secret

type : opaque

data :

- username : dxNlCMnbw=

- password : CqLcmaBcQ=

Note :-

For find username and password, we should pass a command from command line in cluster.

→ echo -n 'username' | base64

OP

[ provides the username ]

→ echo -n 'password' | base64

OP

[ provides the password ]

→ Then we pass that key value in our deployment .yaml file.

Mongo.yaml (cont.) from page : 26 [ Last edit ]

SPEC :

containers :

- name : mongodb  
image : mongo

ports :

- containerPort : 27017

env :

- name : MONGO\_INITDB\_ROOT\_USERNAME

valueFrom :

SecretKeyRef :

name : mongodb-secret

key : Username

- name : MONGO\_INITDB\_ROOT\_PASSWORD

valueFrom :

SecretKeyRef :

name : mongodb-secret

key : Password

→ In this way we can secure our credential in our .yaml deployment file.

## Note :-

If you want to add another file description for deployment of an another application in a single file, then we should use (---) after ending of my file.

- multiple documents in 1 file Possible
- So, for our "Services" file creation, we can use this services connection code in our existing "mongo.yaml" file, instead of using another file for service.

```
apiVersion: v1
kind: Services
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

## Run :-

- ① \$ kubectl apply -f mongo.yaml
- mongodb-service created

## Command

② kubectl get service

Name	Type	Cluster-IP	External IP	Port(s)	Age
kubernetes	clusterIP	10.96.0.1	<none>	443/TCP	36m
mongodb-service	clusterIP	10.96.86.105	<none>	27017/TCP	51s

③ For see all details of our "service".

↳ kubectl describe service mongodb-service

O/P :-

Selector : app=mongodb

Type : clusterIP

IP : 10.96.86.105

Port : <unset> 27017/TCP

Target Port : 27017/TCP

End Points : 172.17.0.6:27017

Session Affinity : None

Events : <none>

④ So, for accessing our application through a browser, we should put an external - services to the service section.

## Command :-

apiVersion : v1

kind : Service

metadata :

name : mongo-express-service

Spec :

selector :

app : mongo-express

type : LoadBalancer

ports :

- Protocol : TCP  
Port : 8081  
targetPort : 8081  
nodePort : 30000

\* So currently we have 3 services run in our cluster.

- ① kubernetes
- ② mongo-express-service (Load balancer)
- ③ mongodb - service .

SO, what is the total process, that, how we able to access the Mongo Express app.

Ans =

