34ii

(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 Stateful set - 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 managed Containerized application.
-> It is used to managed different application, in a thousands of container.
-> This may be a VM, on-premises or in cloud or in hybrid machine.

* what problems does K8s solve?

ans:-

(*) The need for a container orchestration tool?

→ It is used for containing microservices appl" instead of containing monolothic appl".

→ It is used to increased use of - monolothic containers.

(*) what features do orchistration tools offer:-

ans:-

→ It provides high availability or no-downtime.

→ Scalability or high Performance.

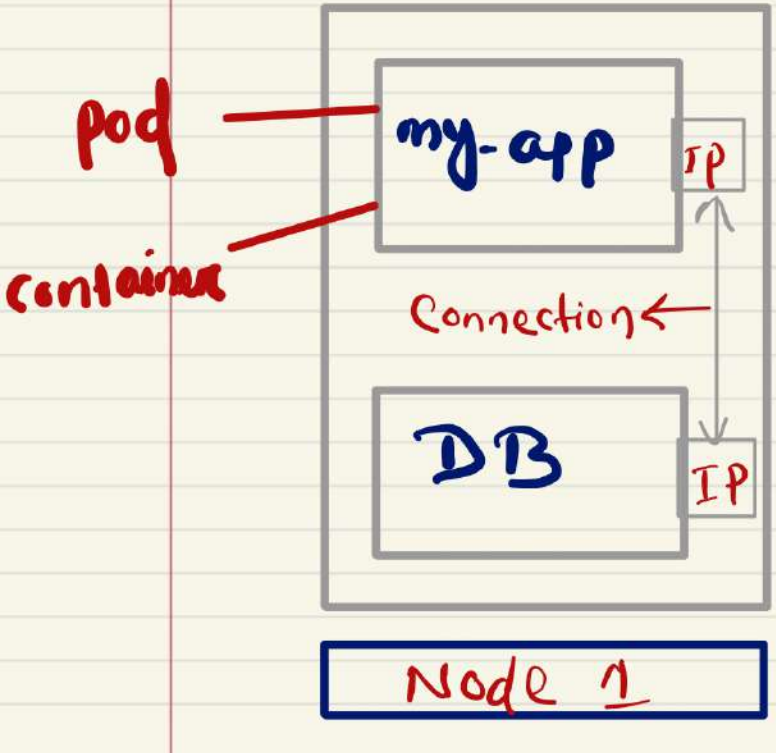
→ In an disaster recover situation - we can take backup and restore.

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 it's own IP address. (Internal one).
-> They use this IP address to communicate.

ConfigMap :-

External configuration of your appl^n.

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.

Imp. (*) Do you face any problem/downtime in your application inside in your k8s cluster.

Ans:- yes, I do have faced multipule problem or 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 stoped.
→ So for overcome this situation, I used replication setting to replicate my Pod in a different one.

→ It means, I do replicate my all pods which contains my all application and run it in a separate node.

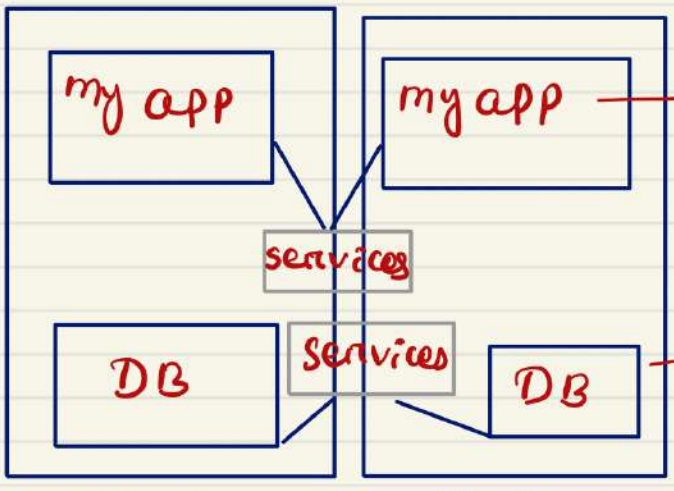
→ Both node's pod connected with same services.

→ Services are two functionalities.

→ Permanenet IP

→ It also is a load Balancer.

→



→ This replica create by "Deployment" techniques.

→ The database replicas create by "statefulset" techniques.

→ For creating extra pods inside in another Node, we used an another components called "Deployement".

→ In "deployement", we don't need to create separate pod for replicate pods in an another Node.

Deployment :-

→ Blueprint for my-app pods.

→ You create Deployements.

→ Abstraction of pods.

Note

→ So now if your one application inside in your one node goes down, Then our application doesn't goes 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 Deployement 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^{n} - Pod like Database.

→ Statefulset applications are like :-

Database

↳ mongo DB
mySQL
Cosmos DB and etc.

Note :-

→ "Deployment" is for Stateless APPs.

→ "statefulset" is for Stateful APPs or Databases.

Imp.

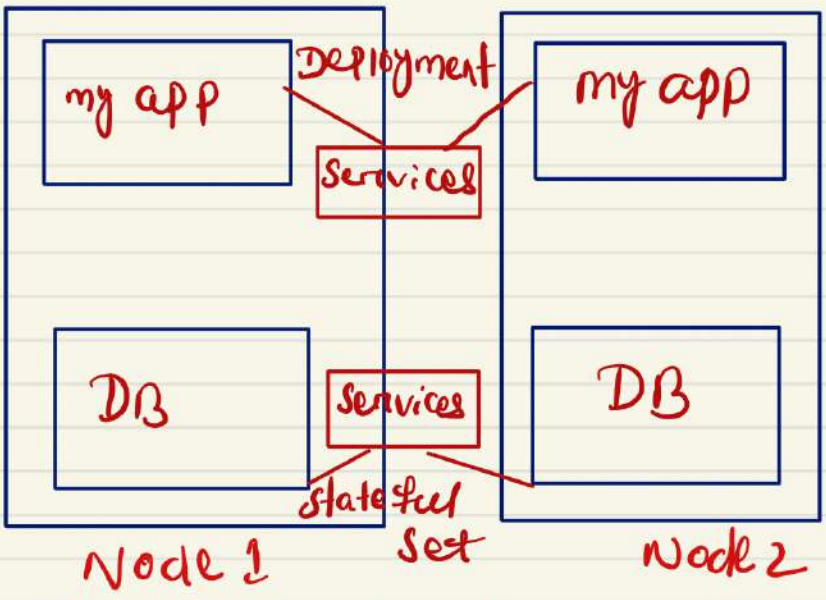
→ Hosting statefulset application is not easy for replication for Databases.

→ 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 all application running in my Node 1 - it goes down, then also our app1 doesn't goes down, because it replica running inside in our Node 2/3 - Pod.

Main kubernetes Components :=

Scemerazed

Pod :- smallest unit of Node
or
Abstraction of Containers.

Services :=
Entablis communication between Pods.

Ingress := Route traffic between clusters.

Config map := Create external configuration

Secrets := Contains secrets.

Volume := Attach External Volumes or Storage in Database.

Deployement := Create replication of Pods containg our application.

Statefulset := Create replication in our Database.

Kubernetes Architecture :=

→ Kubernetes Architecture mainly based on Node.

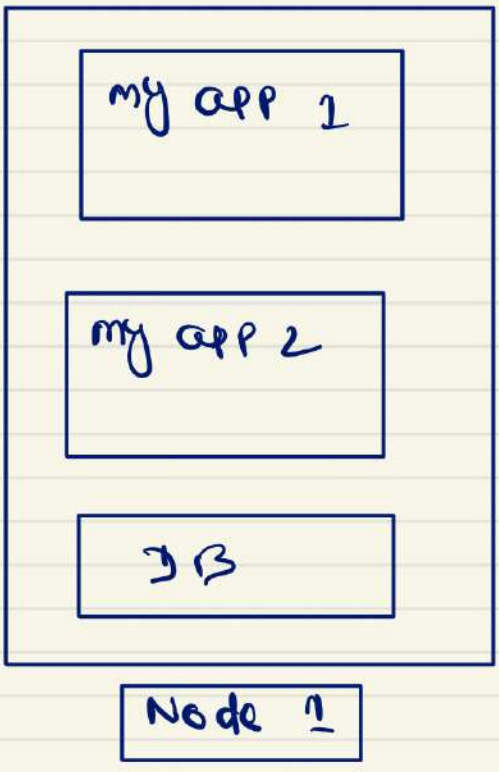
→ We have two types of Node.
  (a) master node
  (b) worker node

→ In a one node, we have multiple Pod.

→ And each Pod contains multiple application.

→ Again clusters are again devided in two methodology
  (1) managed cluster
  (2) un-managed cluster

worker machine in k8s cluster :=

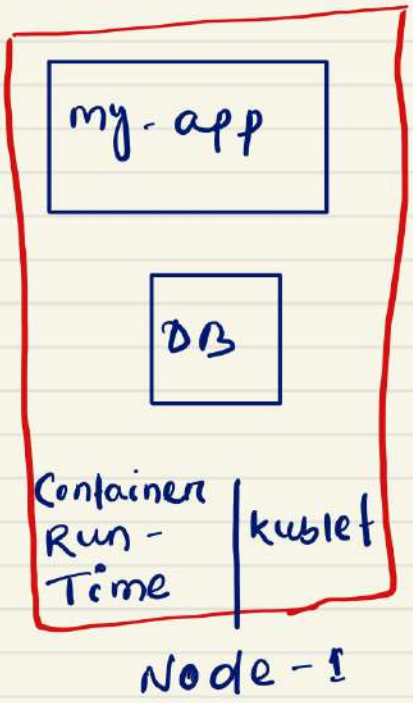


→ Each node has multiple pods in it.

→ 3 process must be installed on every node.

→ Worker nodes do the actual work.

⊛ kubelet and Container Run time :=



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

→ So in a word, in one K8s cluster, we have multipul "Node" running.

→ And inside each "Worcker Node" we have again a no. of pod/Container running.

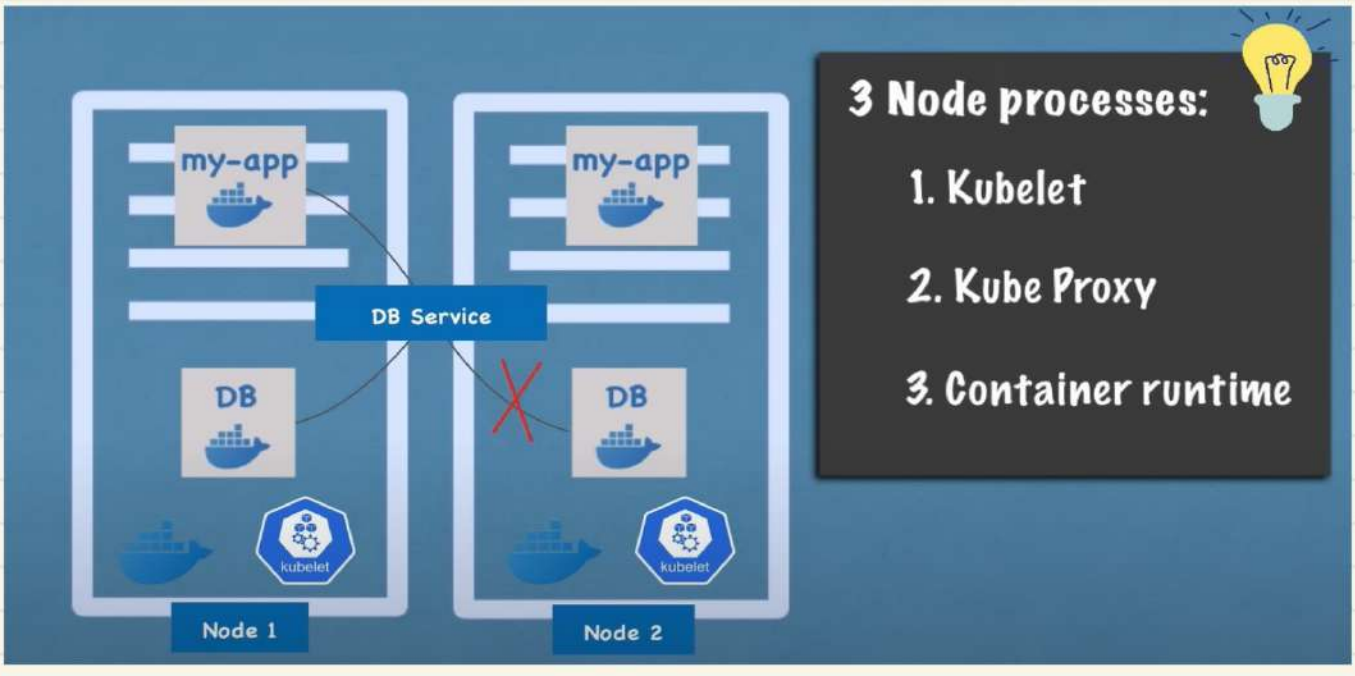
→ So "kubelet" helps here to establish connection between pod and Node.

→ And if we have replica pod running within a another Node, then "Services" is helps to Create Communicaton 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 exfablish in between Pods.

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



→ So, if there in 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 :-

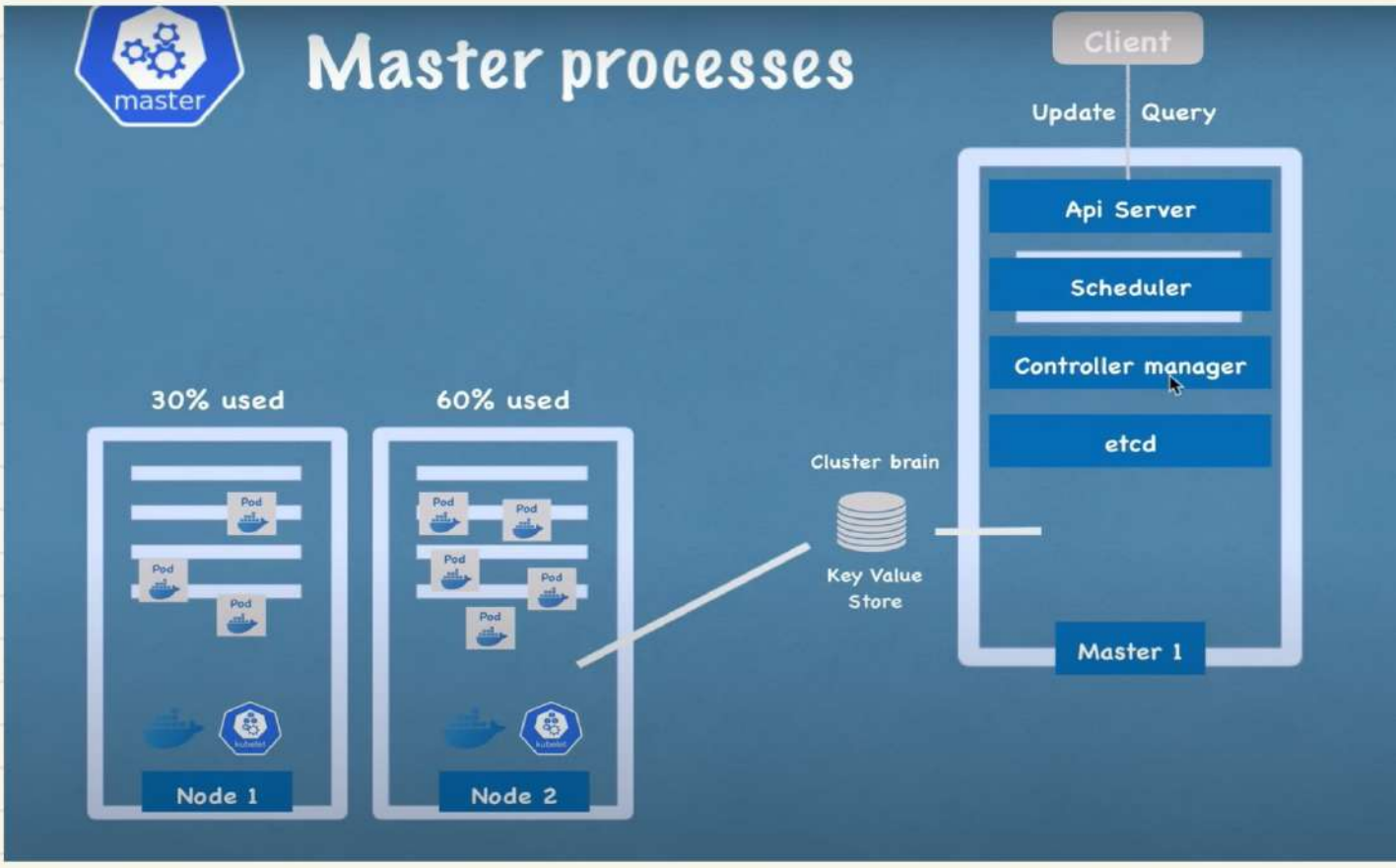
Q:- So, How do you intract with a cluster ?

- ▷ 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

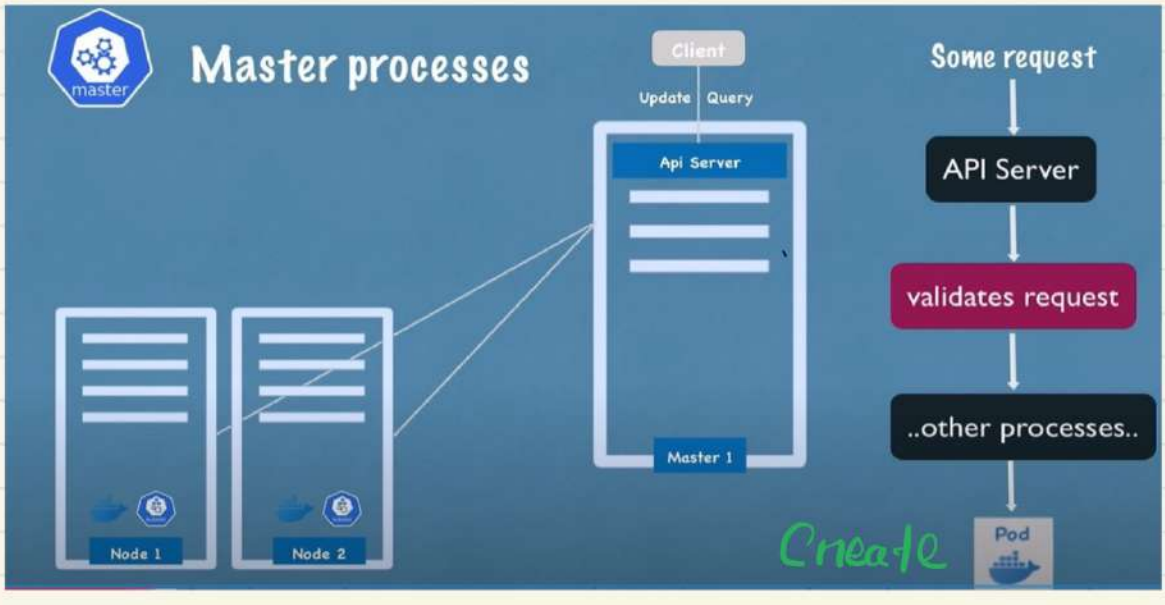
1.) API Server
2.) Scheduler
3.) Controller manager
4.) etcd.

(1.) API Server :=

→ It is the gateway in cluster.

→ It means, it Authenticate, every command before, get installed.

→ "API server" will first authenticate, before creating any pod or new services inside in a node.



(3) Scheduler :-

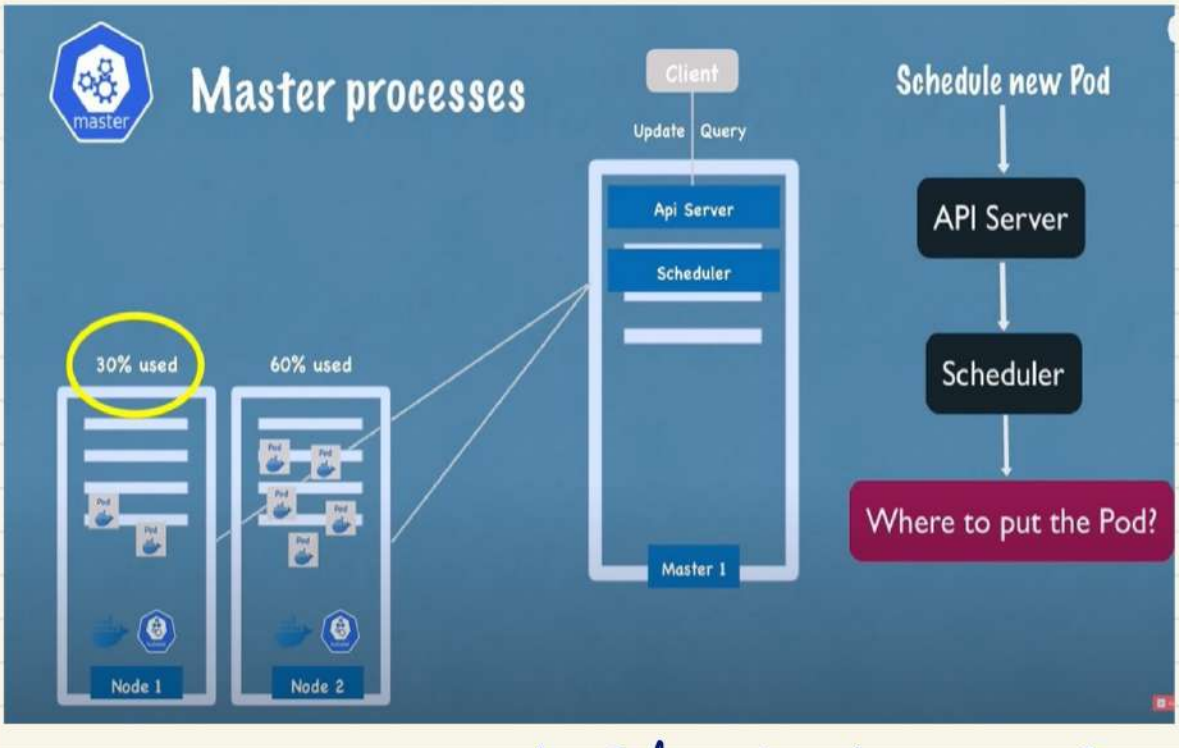
→ Scheduler is the most intelligent tool, who decides, on which Pod, need attention to install any services or any new Pod, for the current time.

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.

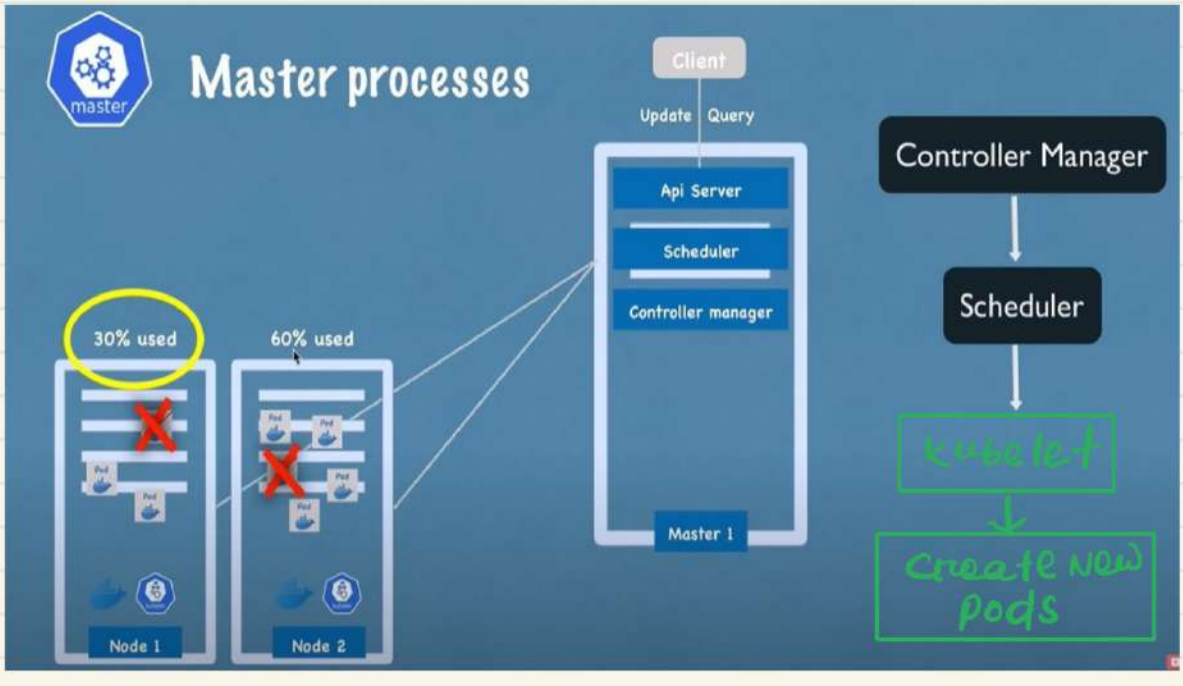


→ So, Scheduler will decides that on which node - new Pod will be scheduled.

(3) controller manager :=

→ If any pod get's down or get's 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 "kubelet" command to replace or transfer all trafic and connection to an another pod.



→ So, finally in one word "controller manager" - detects the stopped "pod" inside ina Node.

→ And after that, it will restart it's replica pod, by using "scheduler", and using "kubelet" command.

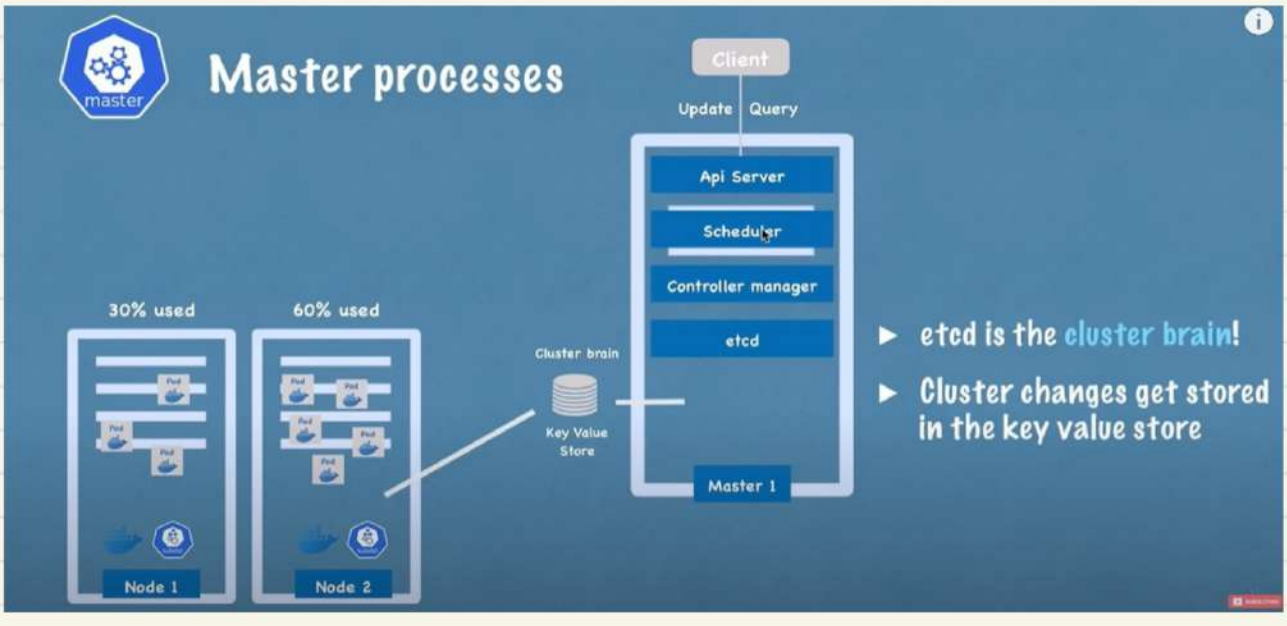
(4) etcd :=

(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 happend inside in a cluster.

(iii) Inside in a cluster, in "etcd", it 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 worker node required more resources. As, it handle more application.

(*) minikube and kubectl setup :=

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

Ⓐ What is minikube?

ans:-

(i) in a production environment, there may be multipul master node and also multipul - worker node.

(ii) So for testing any, environment, it required 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 worker node run inside it,

(iv) Also there in a docker container pre-installed in side it for holding your application.

Note

→ mainly minikube uses 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.

⊛ 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 intract with "API server" first before initiate any command to exicute 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 kubect get pod
③ kubectl get services
④ kubectl get replicaset
⑤ kubectl get deployment

Debuging 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 over 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.

→ Thats 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" appl^n.

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 any thing 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 [Pod name]

Use configuration file for CRUD

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

Imp: 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.

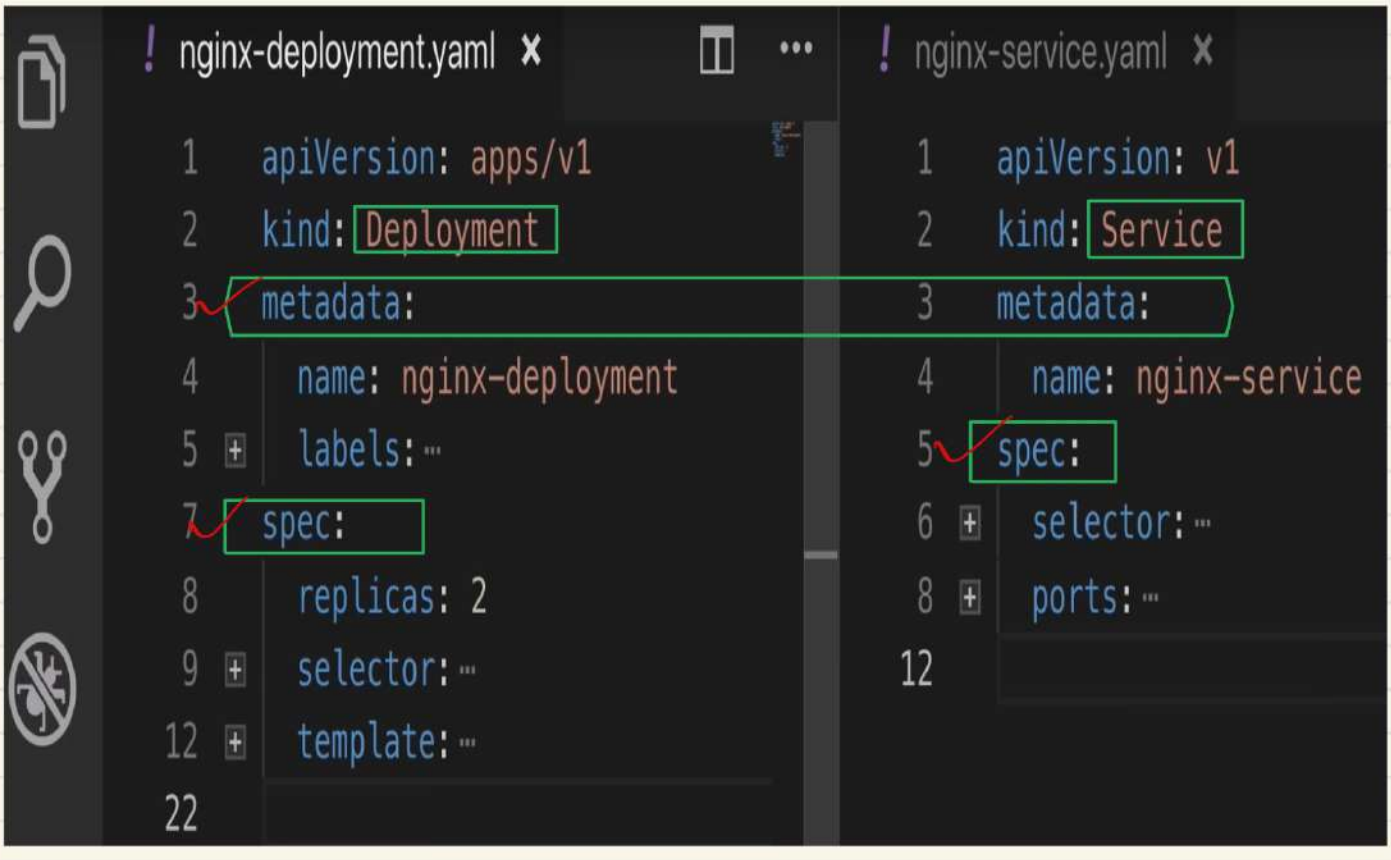
(iii) This is like :=

It connects from "Deployment to Services to-Pods".

* 3 parts of a ".yaml" file.

→ nginx-deployment.yaml

→ nginx-service.yaml



→ 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-rectification process available.

→ It means k8s first compare with "Desired and Actual" Status, and if he found any mis-match in status, it will fix the status.

Desired ? ≠ Actual ?

Desired ? = Actual ? (fixed by k8s)

(*) where does k8s get this status data :=

→ kubernetes has a cluter 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 verry 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 in side 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 pod -o wide.

How k8s generate auto-generated Status file :=

command :-

kubectl get deployment [Pod name] -o yaml

* How to save it through a file :=

kubectl get deployment [Pod name] -o yaml > filename.yaml

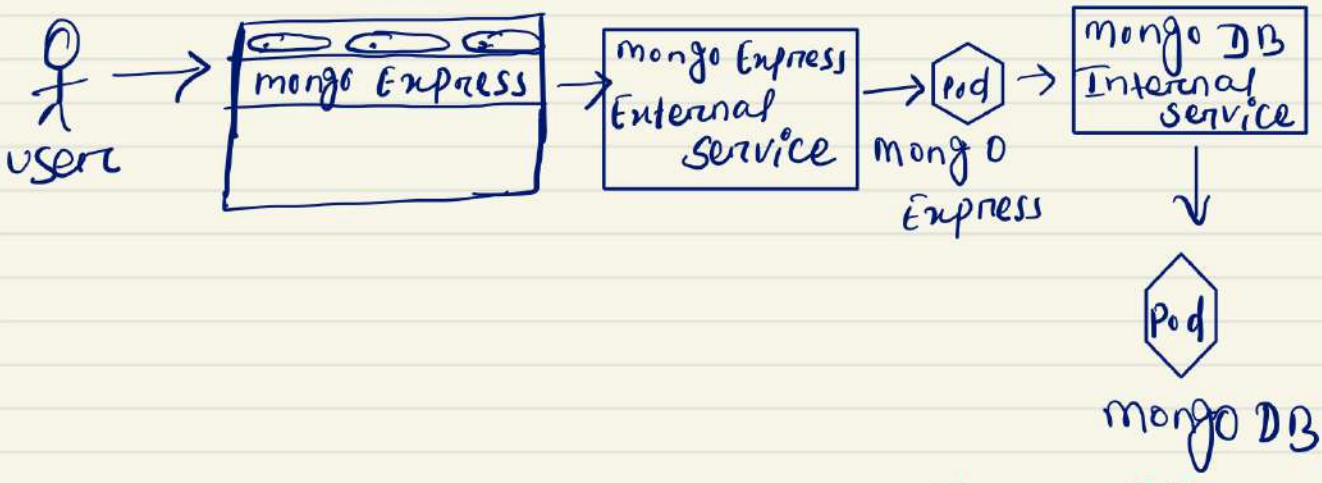
Demo project :=

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
   Selector :
      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:
```

Some changes has made in page → (27)

Note:-

→ For each time our deployement .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: dXNlcm5hbW==
    Password: cGFzcmaBCQ==
```

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

→ echo -n 'username' | base64

o/p

[Provides the username]

→ echo -n 'password' | base64

o/p

[Provides the password]

→ Then we pass that key value in our deployement .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 deployement file.

Note :-

If you want to add another file description for deployement 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

→ monodb - 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".

Command

↳ 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
Endpoints: 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 totall process, that, how we able to acess the mongo Express appl^{n}.

Ans :=