Kubernetes notes

## Kubernetes Installation

Kubernated uses many components. Majority of them being the K8 master nodes, K8 worker nodes, KubeCtl (command line manager for K8) and kube admin (UI based kube admin I suppose).

For the development system, we can deploy minikube – this will give us a single node setup of K8 master and worker.

### Installation commands for Minikube in Ubuntu/Linux.

sudo apt install curl \*\* if you dont have curl installed already.

Minikube needs some VM. Install oracle vm

sudo apt-get install virtualbox

(some popup will be shown askinf to set some machine owner key. press ctl+page down and enter a password and reenter as asked. Reboot the system. continue to boot-ignore any ask for the MOK)

sudo apt-get install virtualbox—ext–pack (this didnt get installed for me)

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

minikube start

This will provide an update on successful startup of the minikube.

## Basic Components:

### Node -

A physical or virtual machine. Nodes can be of 2 types- master node and worker node.

### Master node -

Does not do any actual job of running apps, but it manages how the kube cluster is managed. In a prod setup, there will be minimum 2 master nodes in the cluster.

The machine resources for a master is hence allocated less.

### Worker node -

The one doing actual job of running application in the so called pods. Worker nodes are normally allocated plenty of machine resources as possible.

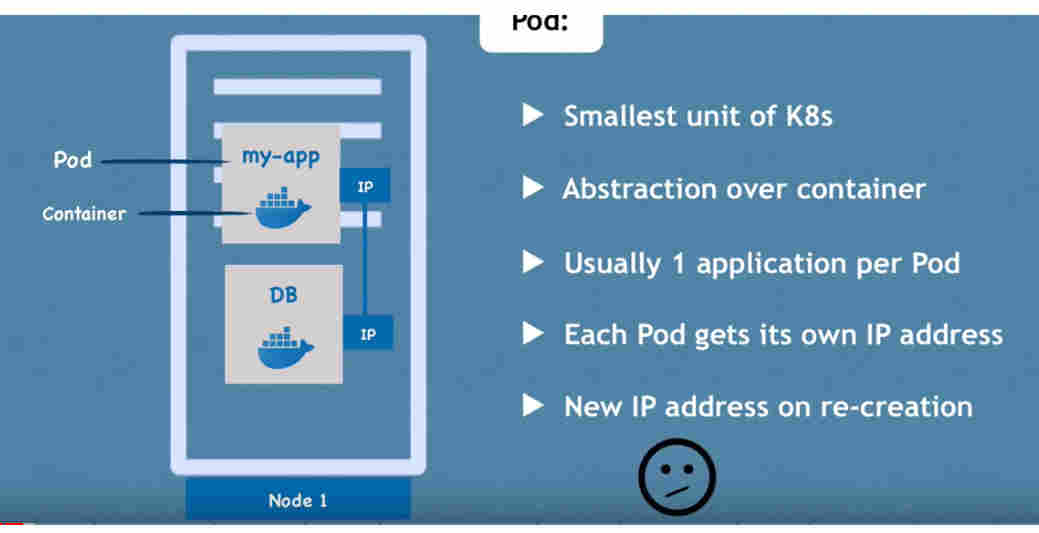
## Kubernetes components

### 1. Pod

Smallest K8s unit. It is an abstraction on the container(eg docker) so that container is not direcly linked to the node in any way. You can always change the container within a pod if you wish to.

Pod is NORMALLY meant to run ONE app container(even though technically we can run multiple containers within a pod- done for some dependent or side apps for the main app).

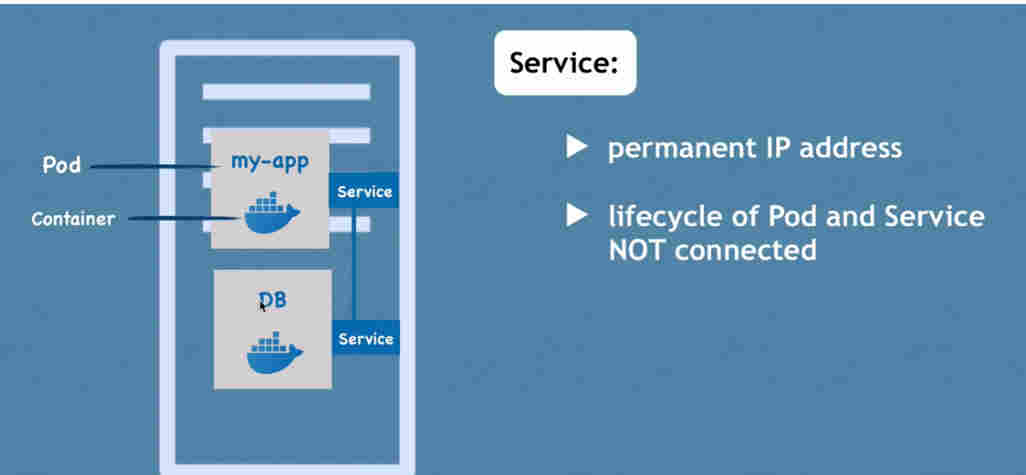
Each pod gets its own IP address for communicating each other. Pods are ephemeral (die easily when node out of resource etc etc). When a new pod is created for a dead pod, it gets new IP address.



### 2. Service

It is a static/permanent IP address that can be attached to a pod. When creating the service, specify it to be Internal or External.

The service is used as Load-Balancer too.

**External Service** – this enables the pod service to be accessible from outside the node.

**Internal Service** – this service can be accessed from within the node.

### 3. Ingress

It enables the service IP address:port to be chnaged to the domain name or something.

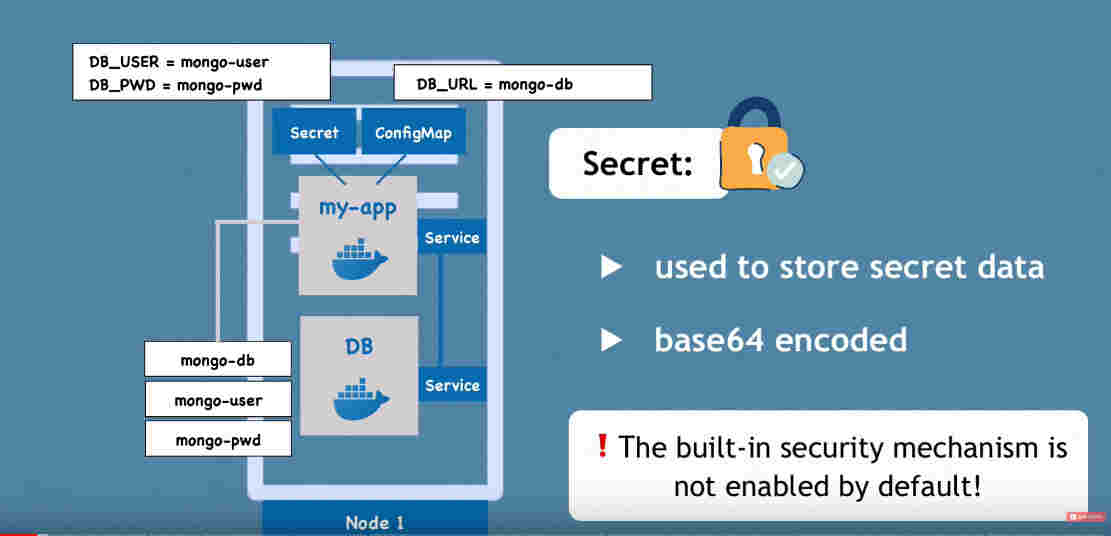
So an external request will come to the Ingress first, and then will go the the Service.

### 4. Secret and ConfigMap

Config map is used to store the global variables to be used in a Pod. This normally saves the db-url etc.

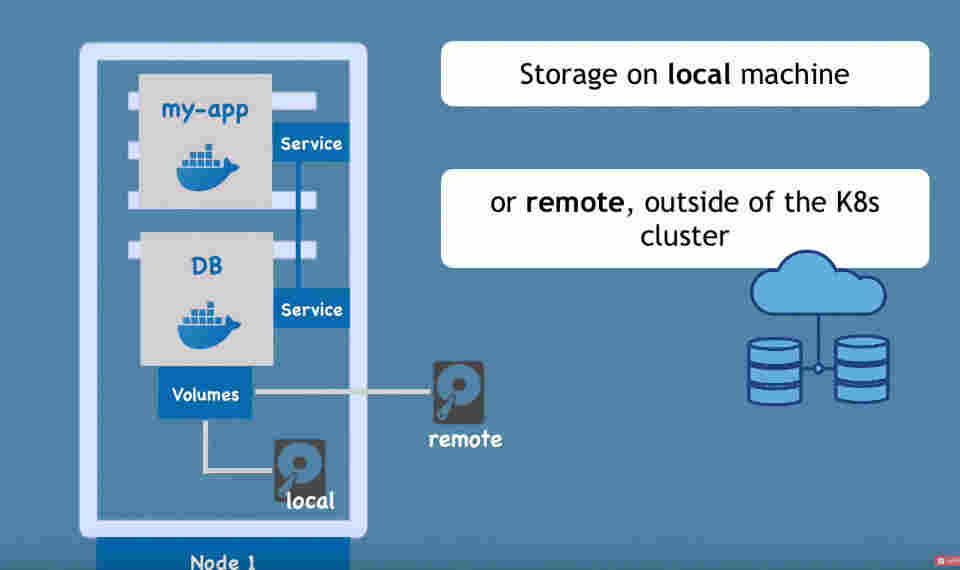
Secret is used to store any credentials like the username and password etc.

config and ConfigMap are linked to the pod.



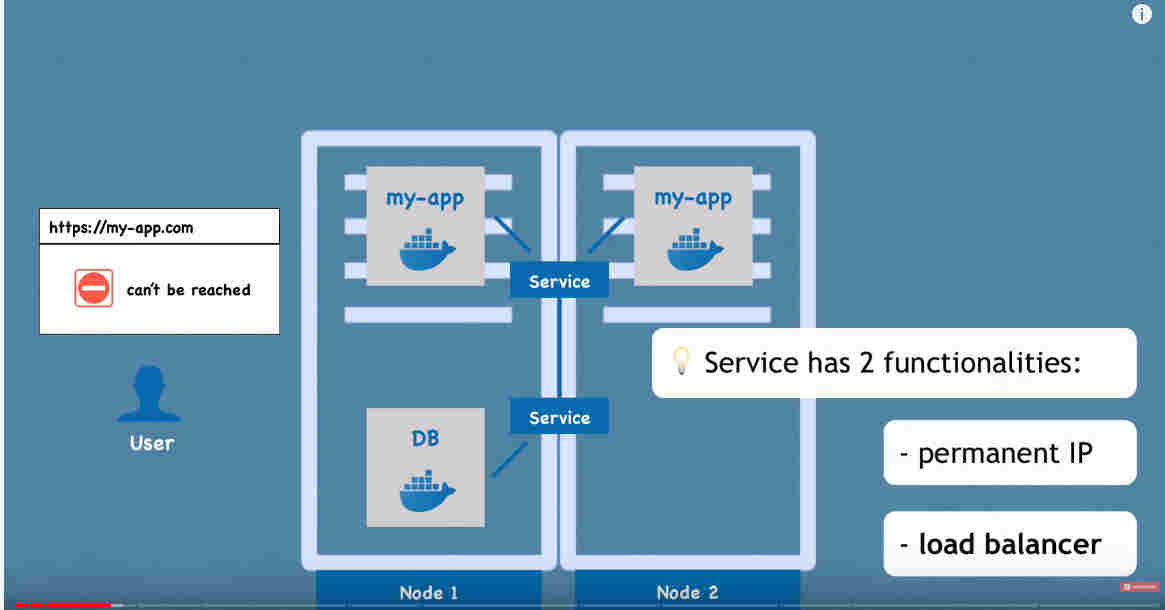
### 5. Volumes

It is used to connect a storage (internal in node or an external/cloud )to the pot.



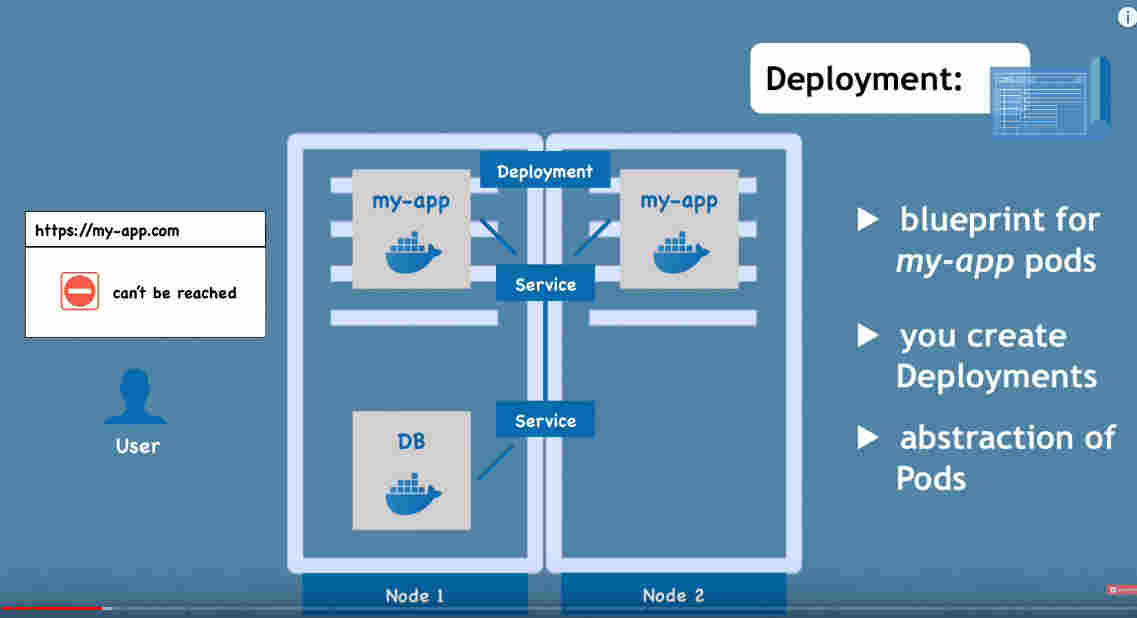
### 6. Replication

In Practice, the pods are always replicated onto multiple nodes. This is done using deployment specification component of K8s. In short, we don’t work with pod, but with deployments.



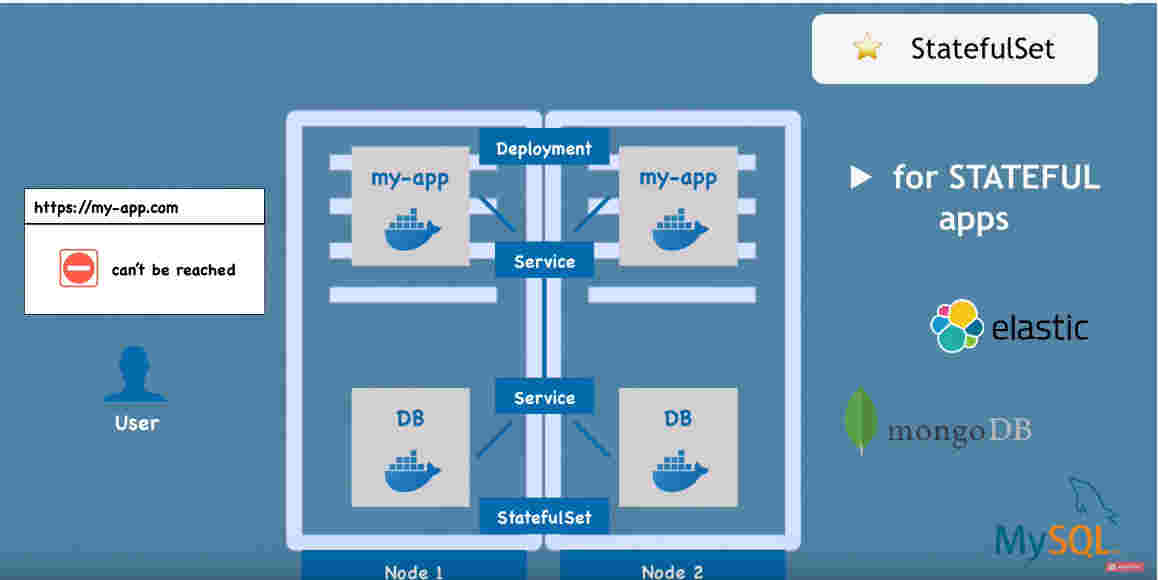
### 7. Deployments

This is a blueprint of all pod specification for our application. This will tell how many pod replicas to be created for each of the pods used(have to specify explicitly) in the application. Any abstraction below the deployment is automatically managed by K8s.

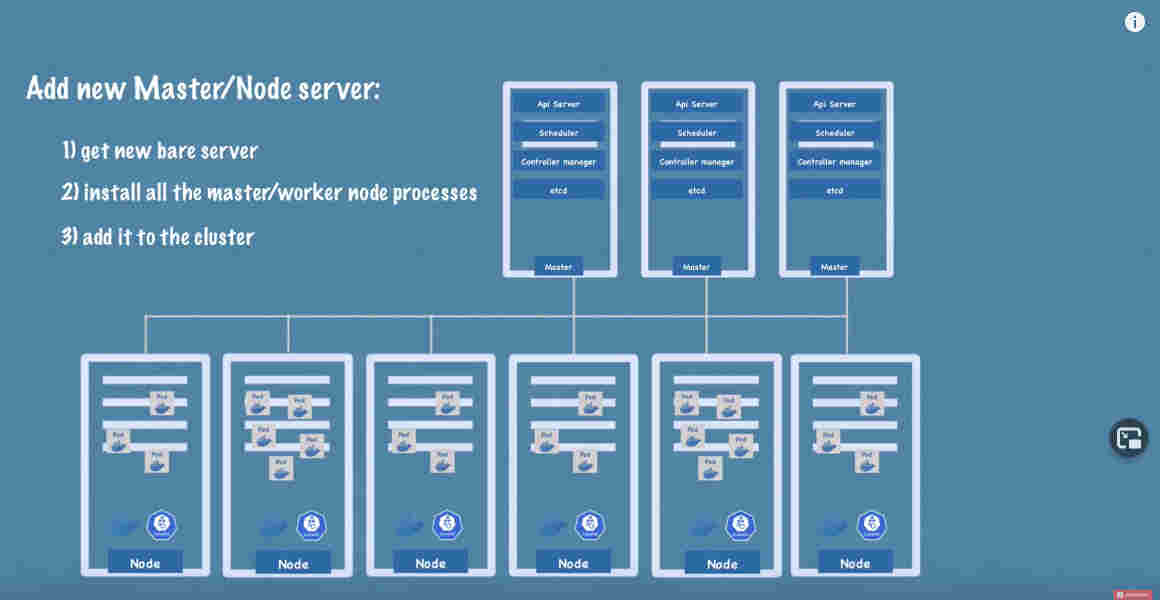


### 8. StatefulSet

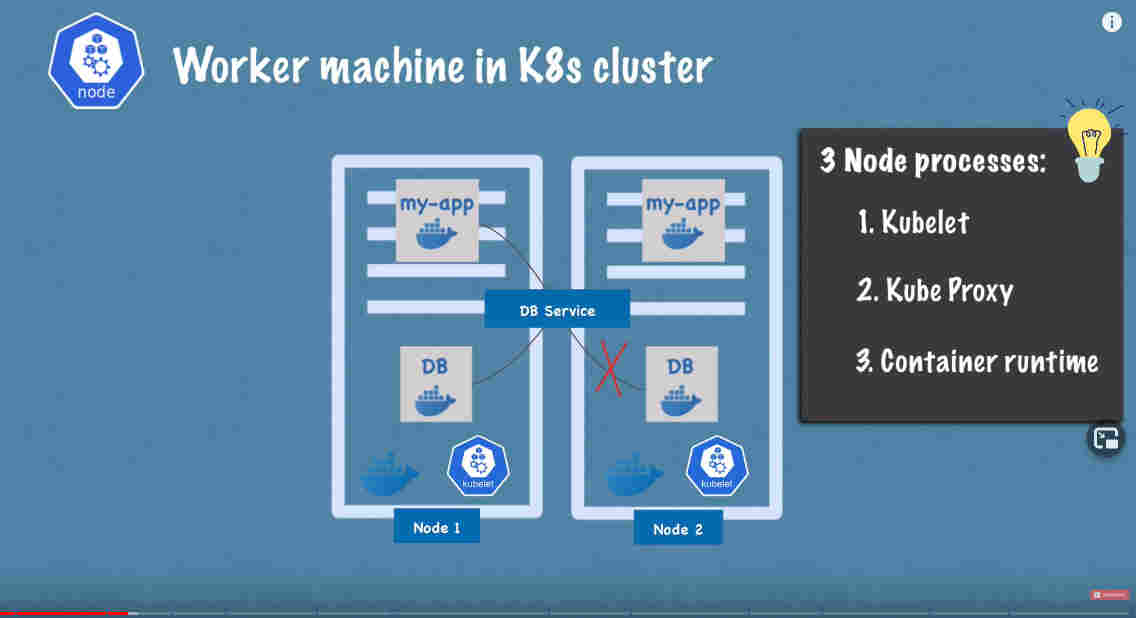
This is used for the deployment of the stateful services like the database. Stateful services do not use the regular Deployment K8s components. But configuring this is very tricky. Its a good practice to host GB outside k8s cluster and keep stateless apps in K8s.



## K8s Architecture



### Worker Node in K8s

A worker node should have 3 processed running so that the pods can be scheduled and managed– kubelet, kubeproxy, container runtime.

### Container runtime –

Any container runtime(like docker) should be running on a node.

### Kubelet

Interface between the container runtime and the NODE. Kubelet will take pod config and startup the pod with a container inside and assign the resources from node to pod.

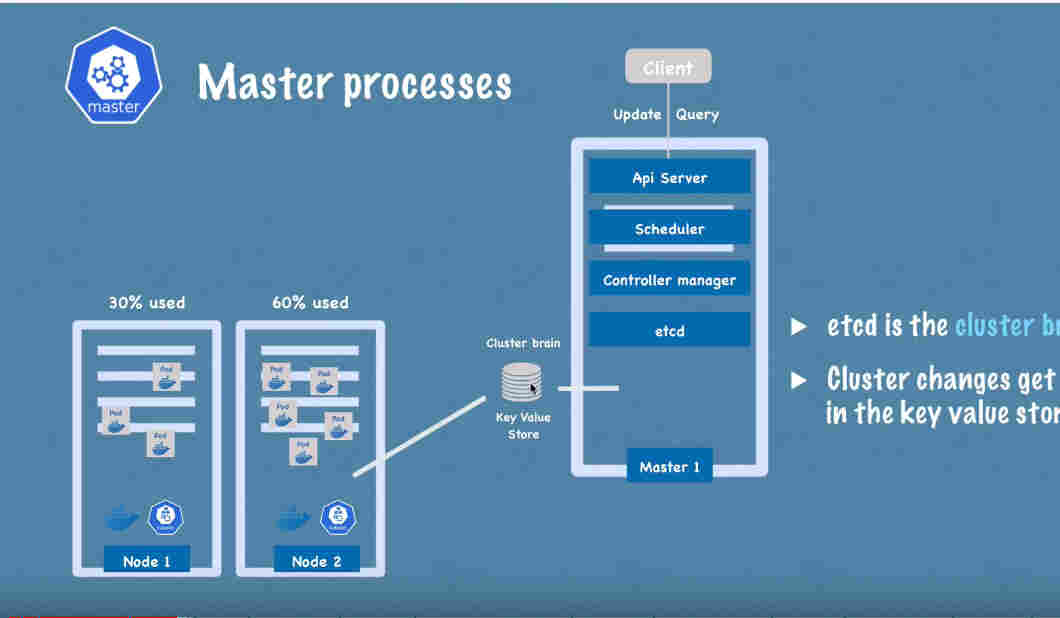
### Kubeproxy

This will forwards the requests coming to the services to the actual pods. This does intelligent routing(requests as contained within a node wherever possible) to reduce network overhead.

### Master node in K8s

Master nodes control the K8s cluster.

4 processes will be running – API server, scheduler, controller manager, etcd



### API Manager

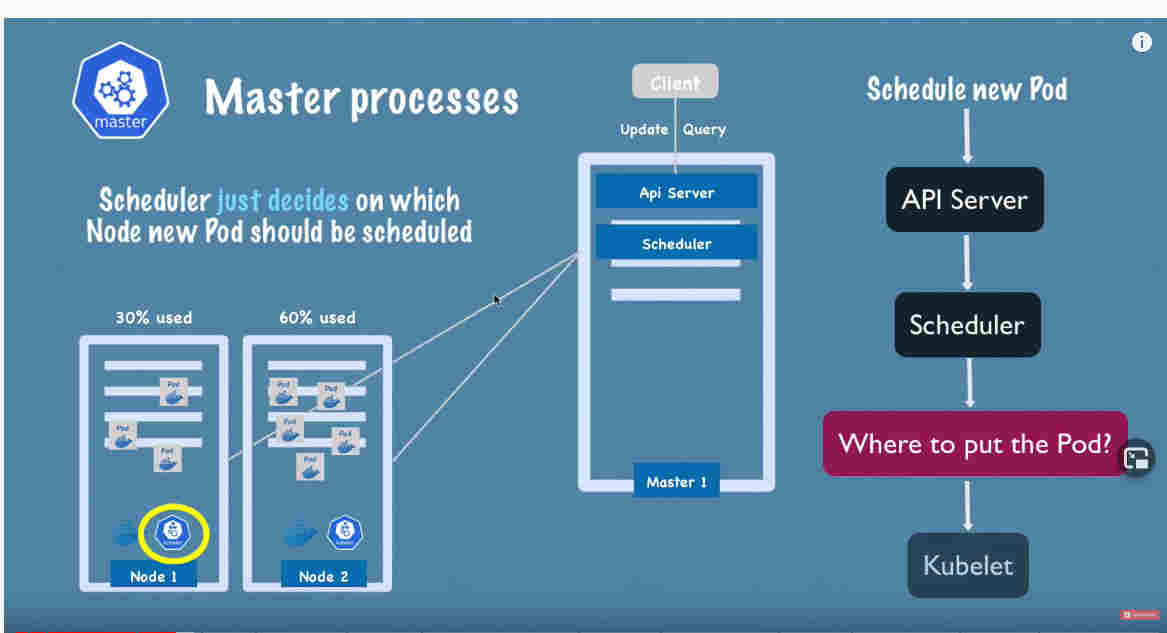
This is the single entry point/cluster gateway to the K8s cluster(for calls like deployment, pod scheduling, health check query etc etc)

Authenticate any query or update request and forward the requests to other processes.

### Scheduler

Takes care of the scheduling of the pods.

Looks at the pod scheduling request resource needs, cross check the resource availability of the resources on all worker nodes(lookup data in etcd) and then decides and forward the request to corresponding kubelet.



### Controller Manager

Monitor Detects the pod state changes like crash.

Inform the scheduler to take action. Scheduler follow the steps just like a new pod scheduling.

Controller Manager – > Scheduler – > Kubelet

### etcd

This is a key value store of whole cluster info, but not the application data.

All changes to cluster (new pod creation, pod dying etc)are recorded in etcd.

## KubeCtl Commands

Official documentation -

https://kubernetes.io/docs/reference/kubectl/overview/

### Install the Kubectl

sudo snap install kubectl (use –classic if throwing error)

### Get the nodes info

kubectl get nodes

minikube kubectl get nodes (this works too if kubectl is not installed on system)

### Get the pod info

kubectl get pod #pods also will work

### view cluster services

kubectl get services

- there is a default service called ‘kubernetes’.

### Creating a deployment (and hence the pod )

In the below example, ***nginx-depl*** is the deployment name.

kubectl create deployment nginx-depl –image=nginx

### Get Replica sets

The replica set configuration is done within the Deployment.

kubectl get replicaset

### Debugging/ Checking pod logs

kubectl logs <<pod name >>

### Describing the pod info

This gives a details pod log info.

kubectl describe pod <<pod name>>

### View the terminal of the a pod

-it stands for Interactive terminal.

Kubectl exec -it [podname] – bin/bash

### Delete a deployment

Kubectl delete deployment [deployment-name]

### Deployment using yaml files

kubectl apply -f [filename.yaml] # -f indicates file

### Get all info of the cluster

kubectl get allocate

### K8s Abstraction Layers



## K8s configuration files

Any config file start with API version and Kind.

API version specify the version.

Kind will tell what kind of config file it is -Deployment, Service etc

after this info the config file has 3 parts

### Metadata

This is info like name and label etc

### Spec

This is the actual specification for the configuration. The parameters will change based on the Kind.

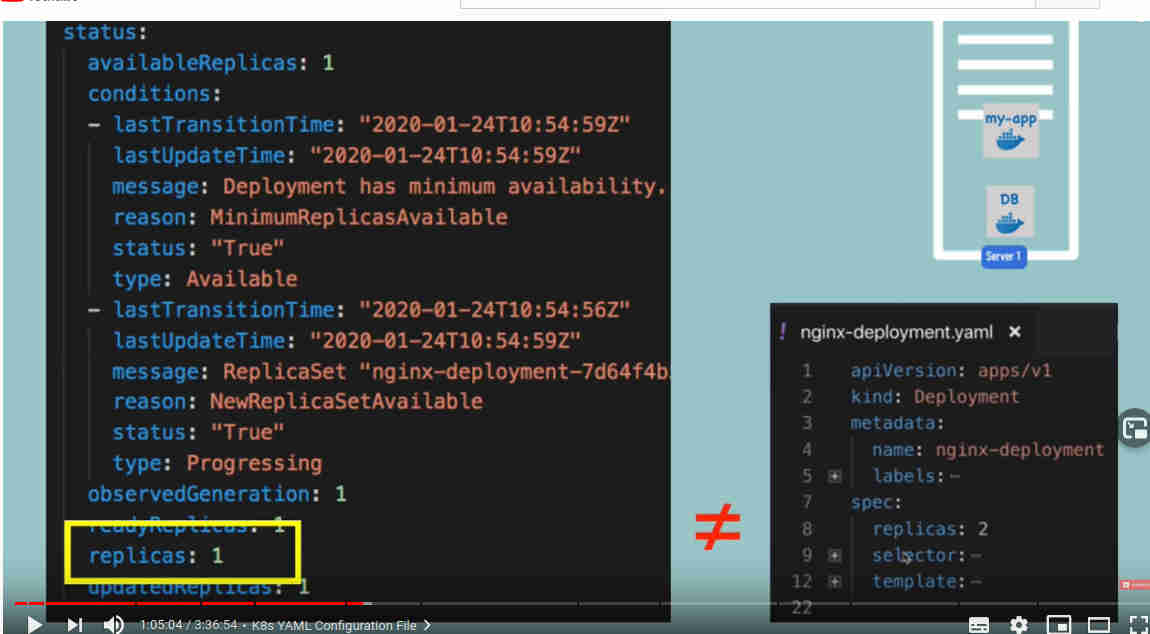
### Status

This is a special section which is NOT done by user but by K8s.

K8s will compare the desired and the actual state continuously and take necessary action.

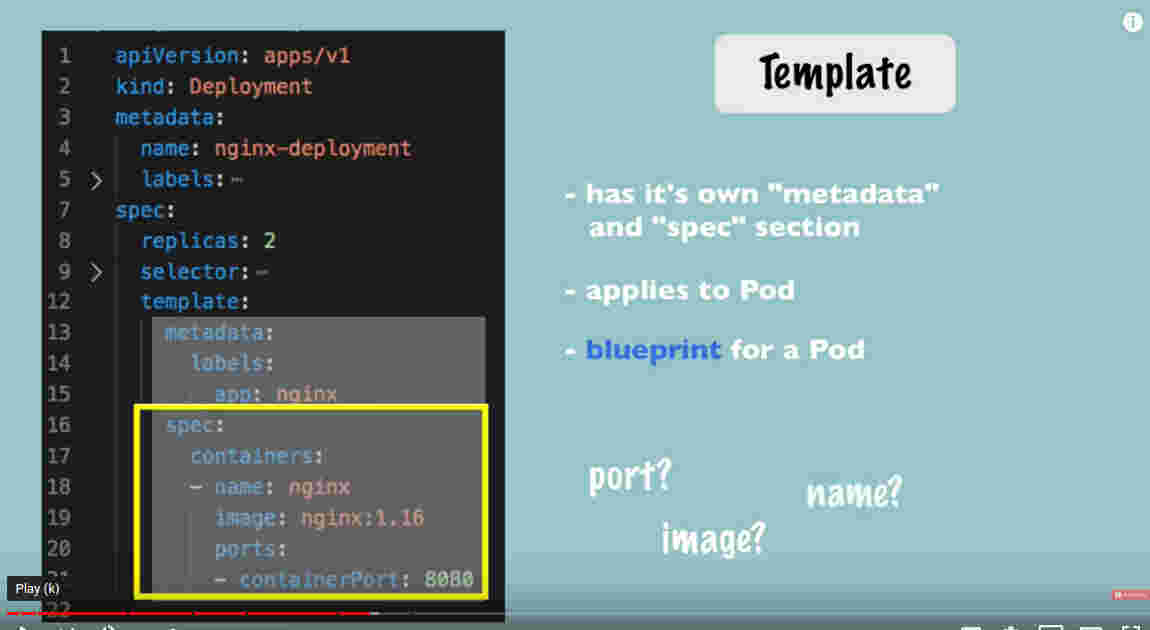
We can view the status section by using get command with out param as yaml

kubectl get deployment [deployment-name] -o yaml



### Spec- Template

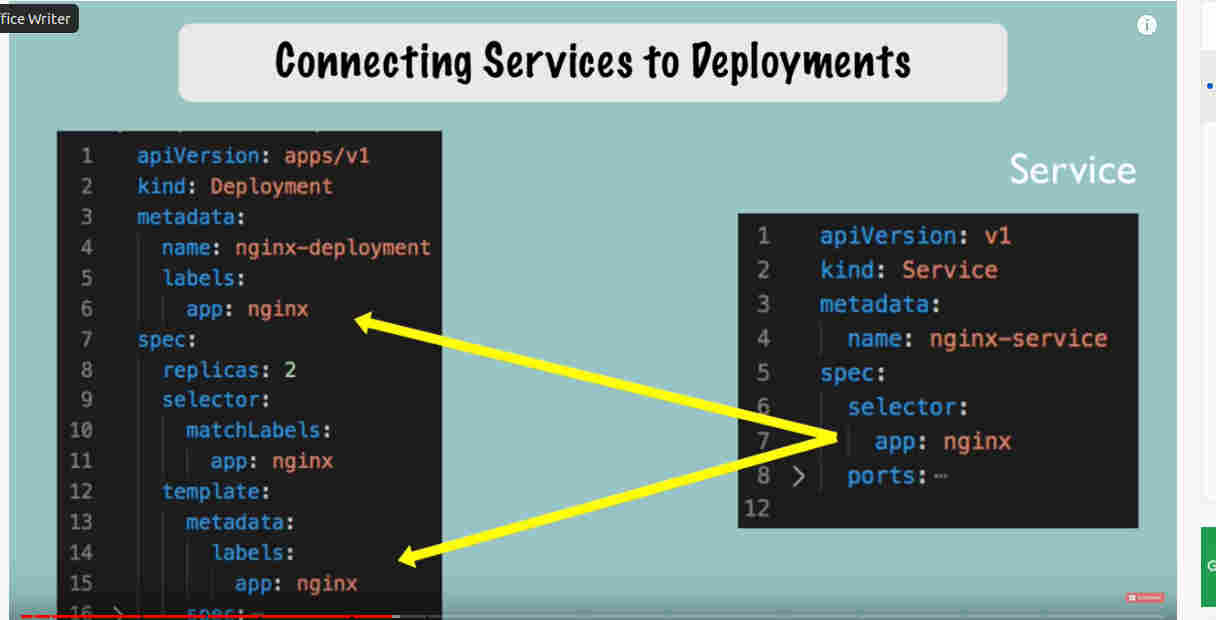
In case of deployments, Spec section will have ***template*** which is used for configuring a pod. Template has its own spec under it.



The labels specific to the deployment and the pod comes from the below sections. This is important to connect the deployment and the corresponding pods.



The Service and the deployment/pods are also linked based on the labels.



### Linking the Service port and pod ports

Target port in service config and the container port in pod spec should match. In the below example, service expose port 80 and route the incoming request to 8080.

