KUBERNETES

- Kubernetes is a cluster manager for docker used for orchestration, service discovery and load balancing
- It is master and node concept
- Kubernetes minions run tasks passed to it by the Kubernetes master
- Kubernetes uses etcd to communicate master and nodes
- Kubernetes having a cluster which it manages the containers in the server when severs get down and run the containers in the servers which was up and auto scales the containers in the server which is up

Components of Kubernetes

1.API Server

2.Etcd

3.kubelet

4.scheduler

5.controller

6.Container runtime

1.API Server

It Acts as the front-end server for kubernetes(API sever is the only way we can talk with kubernetes)

there is command called kubectl by this we can talk to API severs

2.Etcd

Etcd is a distributed reliable key value store to store all data to manage cluster

3.kubelet

Kubelet is an agent to runs on each node of a cluster, Responsible to make sure containers are running in the node.

4.scheduler

Scheduler is to distribute the work across all nodes or containers

5.controller

Main brain behind the kubernetes. If some any containers go down automatically creates new container

6.Container runtime

Container run time in our case it is docker

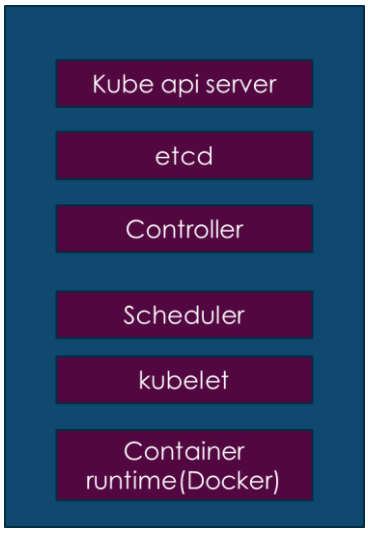
Different ways to Install kubernetes

1.Development environment

2.production environment

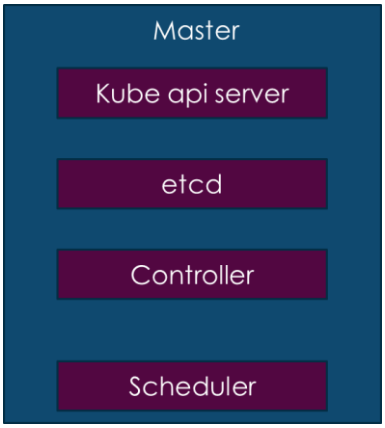
1.Development environment

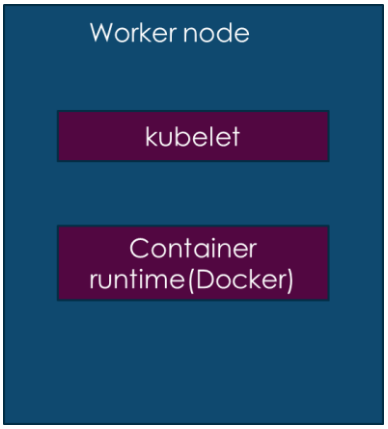
we will use minikube we will use only one server for it



2.production environment

in this we will use two servers master node and slave node





1.installing kubernetes in development environment

Here for kubernetes we need 2 virtual CPU system. In AWS need to launch new instance with type t2.medium

get into the instance and

apt-get update

sudo apt-get install -y conntrack

-First, we should install conntrack because it is related to network-

https://www.radishlogic.com/kubernetes/running-minikube-in-aws-ec2-ubuntu/

-here in this link, we have all instructions to install minikube-

curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl

-it will download & install kubectl which it talks to API server and needed to copy both lines at once and paste in terminal-

chmod +x ./kubectl

-it will give the execute permission-

sudo mv ./kubectl /usr/local/bin/kubectl

-it will move kubectl to bin folder-

sudo apt-get update

sudo apt-get install docker.io

-It will install docker-

curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/

-it will download the minikube-

minikube start --vm-driver=none

-it will start mini cube-

minikube status

There are some kubernetes objects they are

PODS, REPLICASET, SERVICE, DEPLOYEMENT

- POD

pod contains container, we can create more containers in the pod but practically we use one container in one pod
how to create a pod?
pod is created by the yml script,
vi pod.yml

```yaml
---
  apiVersion: v1

  kind: Pod

  metadata:
    name: nodeapp-pod
    labels:
      app: nodeapp

  spec:
    containers:
      - name: weguide
        image: httpd
```

-this is the example for pod.yml

here first API version is to implement version of it-

Kind	Version
POD	v1
Service	v1
Replicaset	apps/v1
Deployment	Apps/v1

kind: it is specifying what type of object I am creating

metadata: it is related to the given object and specify pod name and label as application name

spec: it will create the container with the image name and which image. here httpd is Apache

minikube start

kubectl create -f pod.yml

-it will create or execute the pod-

kubectl get nodes

-it will list the nodes-

kubectl get all

kubectl get pods

-To check the pods-

kubectl describe pod nodeapp-pod

-Detailed information about pod-

kubectl delete pods nodeapp-pod

-It will delete the specific pod-

kubectl delete pod --all

-it will delete all the pods-

kubectl get pods -o wide

-we will get not Ip address and pod Ip address and shows how many times it is restarted-

- repllcaset
  Practically we will not have a single pod. in repllcaset we can use many replicas or pods

```yaml
---
  apiVersion: apps/v1
  kind: ReplicaSet
  metadata:
    name: nodeapp-replicaset
    labels:
      app: nodeapp
      type: front-end
  spec:
    template:
      metadata:
        name: nodeapp-pod
        labels:
        app: nodeapp
      spec:
        containers:
          - name: weguide
            image: httpd
    replicas: 3
    selector:
      matchLabels:
        app: nodeapp
```

-this is the example of replicaset.yml

here everything is same except replicas and sector

here replicas meaning 3 nodes will be running

and sector meaning we will give node to which the similar node should be up when it gets down

- Deployment

It is process which it will create revision for the pods or nodes because if any new version releases the pods should not get down and here in deployment it will create a new revision and replace the new version by one-by-one pod and it will not down all pods at a time so the sever will not get down
and here we can rollout to previous version by undo
and here if new version of node is up and old version will get down

here in deployment.yml script we can use same template like repllcaset, we should only replace the word repllcaset to deployment everything is same

kubectl create -f deployment.yml

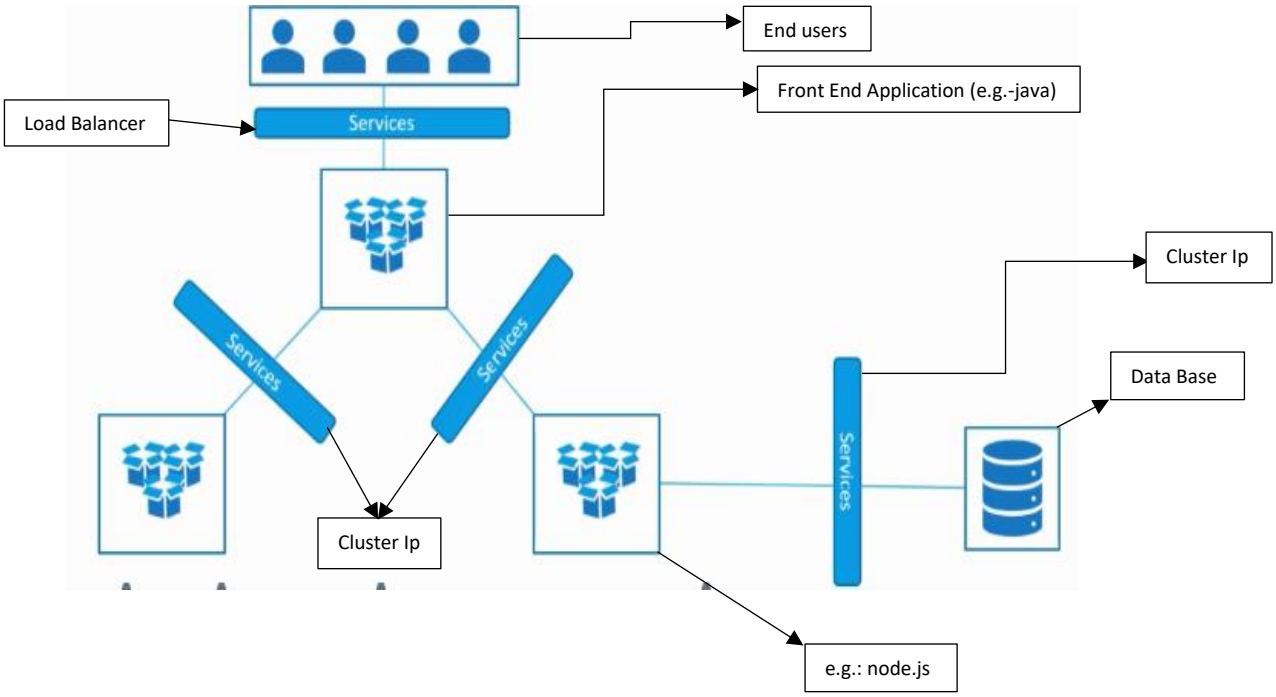
kubectl rollout history <name of the deployment>
-it will show the history of revisions-
kubectl apply -f deployment.yml --record=true
-here this command is used only for version, first we need to create the node and after to change the version of the application we have to create new yml or edit the yml with different version and apply this command. -
kubectl rollout undo <deployment name>
-here this will take back to previous version by specifying the name of the specific deployment-

- Services

A service is like networking how we will be communicating will be done.

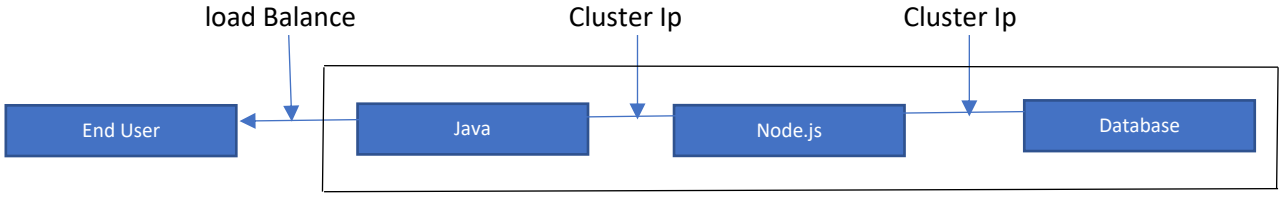
here we have three services:

i. Node port
ii. Cluster Ip
iii. Load Balancer



Here difference between the three services is

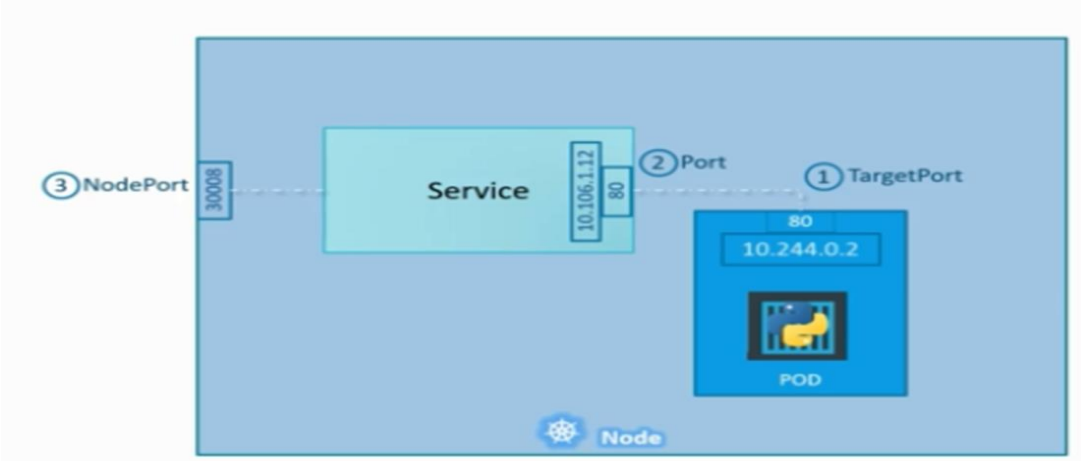
1) Node Port – Internal pod make it available for the node (it is to communicate with single node)
2) Cluster Ip – To enable communication between different nodes (like front end back end etc.), here it will communicate only internally with in the cluster with the specify application
3) Load Balancer – It is to communicate with the front-end Application and end user



here end user can talk to frontend application like java through load balancer itself i.e., load balancer is only can talk from outside

here java should talk to node.js and node.js should talk to data base here internally cluster Ip should be used

Node port:



Node port range will be 30000-32767

vi service.yml

```
---
   apiVersion: v1

   kind: Service

   metadata:
      name: nodeapp-service

   spec:
      type: NodePort
      ports:
      - targetPort: 80
         port: 80
         nodePort: 30008
      selector:
      app: nodeapp
```

kubectl create -f service.yml