# Kubernetes

**1) What is Kubernetes**

- Orchestration Platform
- To manage containers
- Developed by Google using Go language
- Google donated K8S to CNCF
- K8S first version released in 2015
- It is free & Open source

**2) Docker Swarm Vs K8S**

- Docker Swarm doesn't have Auto Scaling (Scaling is manual process)
- K8S supports Auto Scaling
- For Production deployments K8S is highly recommended
- Kubernetes is replacement for Docker Swarm

**3) What is Cluster**

- Group Of Servers
- Master Node(s)
- Worker Node(s)
- DevOps Enginner / Developer will give the task to K8S Master Node
- Master Node will manage worker nodes
- Master Node will schedule tasks to worker nodes
- Our containers will be created in Worker Nodes

**4) Kubernetes Architecture**

- Control Plane / Master Node / Manager Node
    - ▪ Api Server
    - ▪ Schedular
    - ▪ Control Manager
    - ▪ ETCD
- Worker Node (s)
    - ▪ Pods
    - ▪ Containers
    - ▪ Kubelet
    - ▪ Kube Proxy
    - ▪ Docker Runtime

**5) How to communicate with K8S control plane ?**

1)Kubectl (CLI tool)  2)Web UI Dashboard

## Kubernetes Architecture Components

- API Server : It is responsible to handle incoming requests of Control Plane
- Etcd : It is internal database in K8S cluster, API Server will store requests / tasks info in ETCD
- Schedular : It is responsible to schedule pending tasks available in ETCD. It will decide in which worker node our task should execute. Schedular will decide that by communicating with Kubelet.
- Kubelet : It is a worker node agent. It will maintain all the information related to Worker Node.
- Conroller-Manager : After scheduling completed, Controller-Manager will manage our task execution in worker node
- Kube-Proxy : It will provide network for K8S cluster communication (Master Node <---> Worker Nodes)
- Docker Engine : To run our containers Docker Engine is required. Containers will be created in Worker Nodes.
- Container : It is run time instance of our application
- POD : It is a smallest building block that we will create in k8s to run our containers.

## Kubernetes Cluster Setup

1) Self Managed Cluster ( We will create our own cluster )
   a. Mini Kube ( Single Node Cluster )
   b. Kubeadm (Multi Node Cluster)
2) Provider Managed Cluster (Cloud Provide will give read made cluster) ---> Charges applies
   a. AWS EKS
   b. Azure AKS
   c. GCP GKE

## Kubernetes Components

1) Pods
2) Services
3) Namespaces
4) ReplicationController
5) ReplicaSet
6) DaemonSet
7) Deployments
8) StatefulSet
9) K8S Volumes
10) ConfigMap & Secrets
11) Ingress Controller
12) K8S Web Dashboard
13) RBAC (Role Based Access in K8S)

14) HELM Charts (Package Manager)
15) Grafana & Promethues (Monitoring Tools)
16) ELK Stack (Log Monitoring)
17) EKS Cluster (Provider Managed Cluster - Paid Service)

## PODS

- POD is a smallest building block in k8s cluster
- In K8S, every container will be created inside POD only
- POD always runs on a Node
- POD represents a running process
- POD is a group of one or more containers running on a Node
- Each POD will have unique IP with in the cluster
- We can create K8S pods in 2 ways

1) Interactive Mode  (By using kubectl command directley)

Ex: $ kubectl run --name <pod-name>  image=<image-name> --generator=run-pod/v1

2) Declarative Mode (K8S Manifest YML file)

```
---
apiVersion :
kind:
metadata:
spec:
```

- Once K8S manifest yml is ready then we can execute that using below kubectl command

**$ kubectl apply -f <file-name>**

## Kubernates Sample POD Manifest YML

```
---
apiVersion: v1
kind: Pod
metadata:
  name: javawebapppod
  labels :
    app: javawebapp
```

```
spec:
  containers:
   - name: javawebappcontainer
     image: ashokit/javawebapp
     ports:
        - containerPort: 8080
...
```

```
$ kubectl get pods
$ kubectl apply -f <pod-yml>
$ kubectl get pods
$ kubectl describe pod <pod-name>
```

***** Note: By default PODS are accessible only with in the Cluster, Outside of the Cluster We can't access PODS*******

=> To provide PODS access outside of the cluster we will use 'Kubernetes Service' concept

## K8S Service

- K8S service makes PODs accessible outside of the cluster also
- In K8S we have 3 types of Services
    1) Cluster IP
    2) Node Port
    3) Load Balancer ( Will work only with Provider Managed Cluster - we will learn this in EKS )
- We need to Create k8s service manifest to expose PODS outside the cluster

```
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
spec:
  type: NodePort
  selector:
          app: javawebapp
  Ports:
          - port: 80
            targetPort: 8080
...
```

```
$ kubectl get svc
$ kubectl apply -f <service-manifest-yml>
$ kubectl get svc
```

**Note: NodePort service will map our pod to a random port Number (Ex: 30002)**

-> Enable Node Port in Security Group Inbound Rules

```
$ kubectl describe pod <pod-name>
```

Note: Here we can see in which Node our POD is running

-> We can access our application using below URL

**URL : http://node-ip:node-port/java-web-app/**

## Cluster IP

- It will expose our k8s service on a cluster with one internal ip
- Cluster IP type service is accessible only with in cluster using Cluster IP
- When we access cluster ip, it will redirect the request to POD IP

Note: POD is very short lived object, when pod is re-created POD ip will change hence it is not at all recommended to access pods using pod ips. To expose PODS with in cluster we can use 'Cluster IP' service

Note: ClusterIP service is accessible only with in cluster (can't accessed outside the cluster)

-> To expose POD using service, we will use POD label as a Selector in Service Manifest file like below

```
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
spec:
  type: ClusterIP
  selector:
          app: javawebapp # this is pod label
  Ports:
          - port: 80
            targetPort: 8080
...
```

```
$ kubectl apply -f <svc-manifest-yml>
$ kubectl get svc
```

## Node Port

- Node Port Service is used to expose our PODS outside the cluster also
- When we use NodePort Service we can specify PORT Number, if we don't specify port number then k8s will assign one random port number for our service.

Q) What is the range of NodePort service PORT Number in k8S ?

Ans) 30000 - 32767

```
---
apiVersion: v1
kind: Service
metadata:
 name: javawebappsvc
spec:
  type: NodePort
  selector:
          app: javawebapp
  Ports:
          - port: 80
            targetPort: 8080
            nodePort: 30002
...
```

```
$ kubectl get svc

$ kubectl apply -f <svc-manifest-yml>

$ kubectl get svc

$ kubectl delete service <service-name>
```

Note: Once we expose our POD using NodePort service then we can access our pod outside the cluster also.

```
# Get POD IP and POD Running Node IP

$ kubectl get pod -o wide

#URL To access our application

        http://pod-running-node-public-ip:nodeport/<context-path>/
```

## Comibining Pod manifest and Service Manifest using single YML

```
---
apiVersion: v1
kind: Pod
metadata:
  name: javawebapppod
  labels :
    app: javawebapp
spec:
  containers:
   - name: javawebappcontainer
    image: ashokit/javawebapp
    ports:
      - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: javawebappsvc
```

```
spec:
  type: NodePort
  selector:
    app: javawebapp
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30002
...
```

$ kubectl apply -f <manifest-yml>

## POD Lifecycle

- When we make a request to create a POD then API Server will recieve our request
- API Server will store our POD creation request in ETCD
- Schedular will find un-scheduled pods and it will schedule them in Worker Nodes
- The Node Agent (Kubelet) will see POD Schedule and it will fire Docker Engine
- Docker Engine runs the Container
- The entire POD lifecycle is store in ETCD.

Note : POD is ephemeral ( very short lived object )