**\* \* \* \* \* Kubernetes \* \* \* \* \***

**09-28-k8s-25-Nov-24**

## Git Repo : https://github.com/ashokitschool/kubernetes\_manifest\_yml\_files.git

**=================================================================**

**\* \* \* \* \* \* \* \* \* \* \* \* Kubernetes (K8S) \* \* \* \* \* \* \* \* \* \* \* \***

**=================================================================**

=> It is free and open source s/w

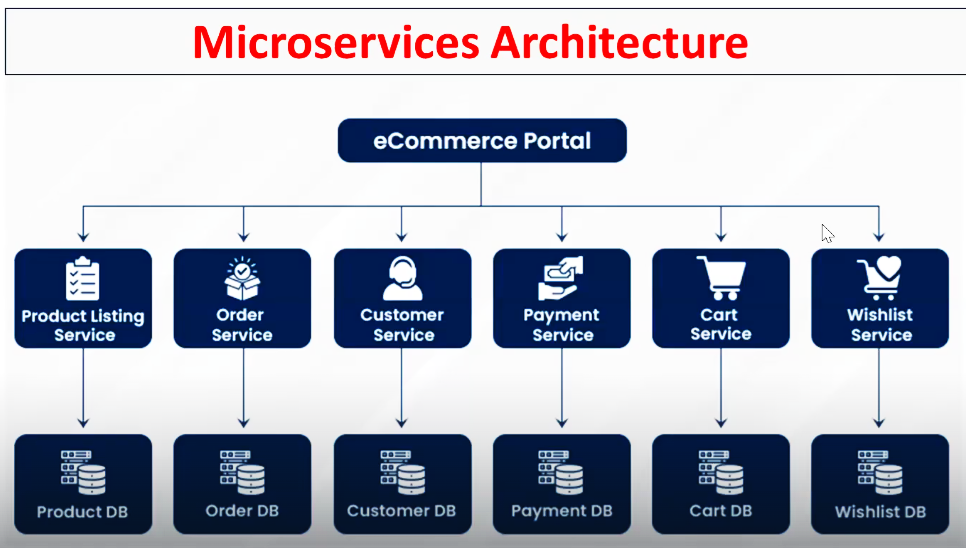
=> K8S developed by google company.

=> K8S provides Orchestration platform.

Note: Orchestration means Management.

=> Using k8s we can manage docker containers

Ex: create, stop, start, scale up, scale down and delete.



**=====================**

**Docker vs Kubernetes**

**=====================**

Docker : containerization platform.

Note: Using docker we can package our "application code + application dependencies" as single unit for execution.

Note: Once docker image got created then k8s work will start.

Kubernetes : Orchestration platform.

**=======================**

**Kubernetes Advantages**

**=======================**

1) Free Of cost : No need purchase

2) Orchestration Platform : Manages our application containers

3) Self Healing : If any container got damaged then it will replace with new container. We can achieve High Availability.

4) Load Balancing : It will distribute incoming requests to all containers in round robbin fashion.

5) Scalability : Increase and decrease containers based on demand

(Auto Scaling)

**=========================**

**Kubernetes Architecture**

**=========================**

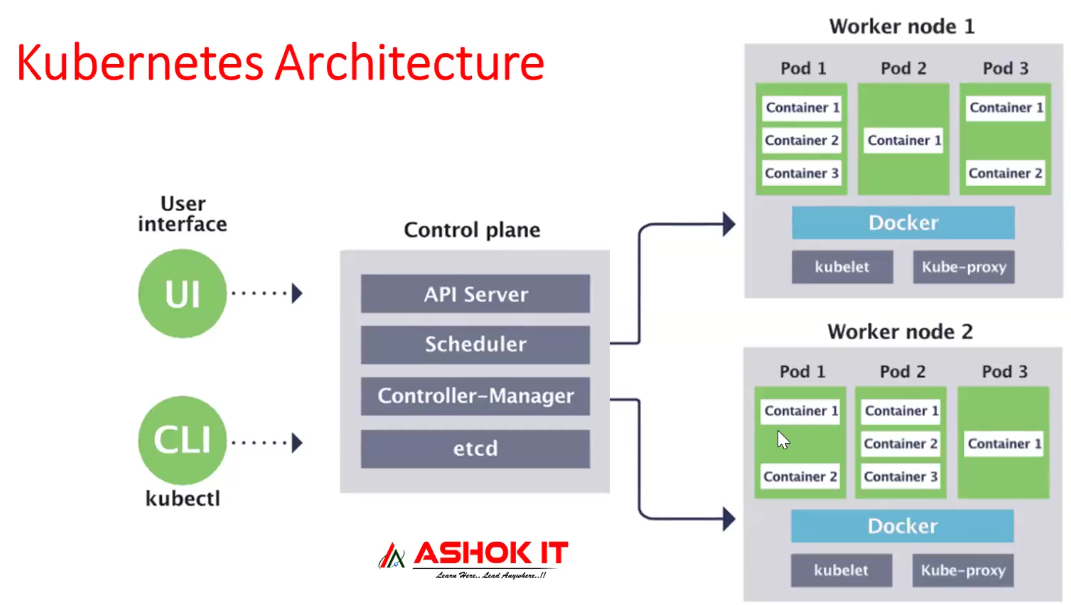
=> K8S will follow cluster architecture.

Cluster : Means Group of servers/machines.

=> In K8S cluster we can see below components

**1) Control Plane / Master Node**

**2) Worker nodes**

****

**=============================**

**K8S Architecture Components**

**=============================**

**0) Kubectl :**

**1) Control Plane**

- API server

- Schedular

- Controller Manager

- ETCD

**2) Worker Node**

- Kubelet

- Kubeproxy

- Docker

- POD

- Container

* To deploy our application using k8s then we need to communicate with control node.
* We will use KUBECTL (CLI) to communicate with control plane.
* "API Server" will recieve the request given by "kubectl" and it will store that request in "ETCD" with pending status.
* "ETCD" is an internal database of k8s cluster.
* "Schedular" will identify pending requests available in ETCD and it will identify worker node to schedule the task.
* Note: Schedular will identify worker node using kubelet.

**Kubelet**

* "Kubelet" is called as Node Agent. It will maintain all the worker node information.
* "Kube Proxy" will provide network for the cluster communication.

**Controller Manager**

* "Controller Manager" will verify all the taks are working as expected or not.
* In Worker Node, "Docker Engine" will be available to run docker container.

**POD**

* In K8s, container will be created inside POD.
* POD is a smallest building block that we can create in k8s cluster to run our applications.
* Note: In K8s, everything will be represented as POD only.
* Pods are used to deploy applications and manage their lifecycle within a Kubernetes environment.

**==========**

**K8S Setup**

**==========**

* We can setup k8s cluster in multiple ways

1) **Self Managed k8S cluster** (We need to setup everything)

a) Mini Kube => Sinle Node Cluster => Only for beginners practice

b) Kubeadm => Multi Node Cluster (HA)

2) **Provider Managed K8S cluster** (ready made)

**a) AWS EKS**

**b) Azure AKS**

**c) GCP GKE**

**d) IBM IKE**

Note: Provider will give ready made cluster for us.

#### Note: Provider Managed Clusters are chargable #####

**========================================**

**Minikube setup in windows for practice**

**========================================**

1) Download and install docker desktop s/w

2) Verify docker installation by opending 'command prompt'

Ex: docker run hello-world

3) Download and install minikube software (.exe file)

4) Open Command prompt and start minikube cluster

Ex:

minikube start --driver=docker

minikube status

kubectl get nodes

kubectl get pods

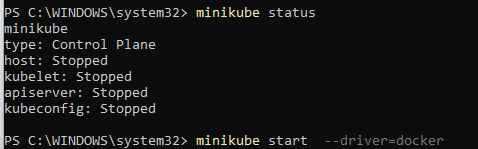
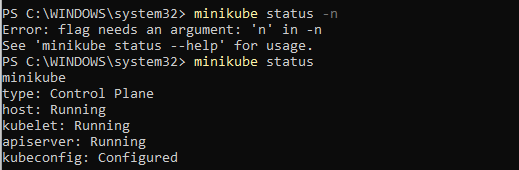
kubectl create deployment nginx-deployment --image=nginx:latest --replicas=3

kubectl get pods

kubectl delete pod <pod-name>

----------------------------------------------------------------------------------------------------------------

**Docker Desktop + minikube setup**

* Download docker desktop and enable all the setting from the youtube .
* While installing docker desktop enable both the ticks and install docker check is docker is running or not .
* Open powershell and run - docker run hello-world
* Install latest minikube for windows and go to the C drive/program files/kubernetes/minikube copy path and set environment varibles .
* Then open powershell with run as administarator .
* Run command – minikube status
* 
* Minikube status if minikube not available then enter below command
* Command - minikube start --driver=docker
* 
* Kubectl is used to communicate with the cluster it is installed by default , by using kubectl we can perform opreations in the cluster .

**Command – kubectl get nodes**

* This command is used to check how many machines are available in our system .

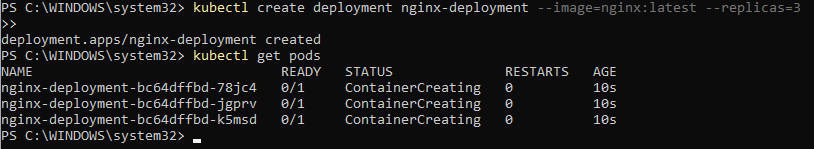
**Command – kubectl get pods**

* Our application running inside a pod only
* In kubernetes everything will be represented as a POD only .

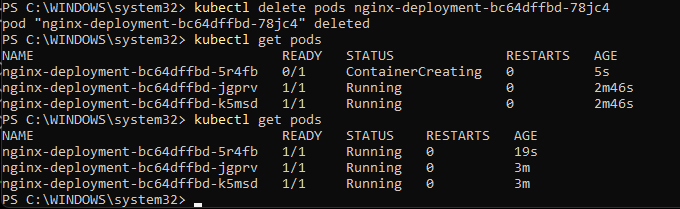
**Command – kubectl get service**

How to create a POD in kubernetes ?

* To create a POD we need docker image
* **kubectl create deployment nginx-deployment --image=nginx:latest --replicas=3**

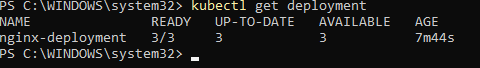


**Command – Kubectl delete pods <POD-NAME>**



* after deleting one pod automatically another POD created
* self healing working properly .

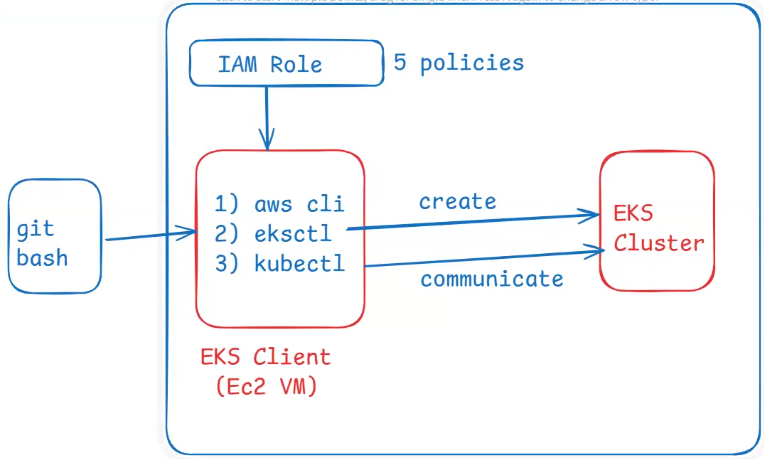
**Command – kubectl get deployment** ( deployment created by us)



**==============**

**AWS EKS Setup 11-28-k8s-27-Nov-24**

**==============**

****

EKS => EKS stands for Elastic Kubernetes Service.

=> EKS is a AWS cloud Provider Managed cluster.

Note: EKS is a paid service in AWS cloud.

=> EKS is the most trusted platform to deploy our applications.

@@ Steps To Setup : https://github.com/ashokitschool/DevOps-Documents/blob/main/05-EKS-Setup.md

Note: After practice don't forget to delete your EKS cluster.

Command – kubectl delete all –all

* it will delete all the the available pods permanently .

**===============**

**What is POD ?**

**===============**

=> POD is a smallest building block in the k8s cluster.

=> Applications will be deployed as PODS in k8s.

Note: To create PODS we will use docker images.

=> To create PODS we will use Manifest YML file.

=> For single docker image we can create multiple PODS also.

=> If we run application with multiple pods then we will get High availability and Load Balancing.

=> PODS count can be increased and decreased based on the demand

(scalability).

**=====================**

**K8S manifest YML**

**=====================**

=> In K8S manifest YML file we will write 4 sections mainley

apiVersion: <version-number>

kind: <resource-type>

metadata: <name>

spec: <container>

===================

POD MANIFEST YML

===================

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javac1

image: ashokit/javawebapp

ports:

- containerPort: 8080

...

Note: Save above content in .yml file

# execute manifest yml

kubectl apply -f <manifest-yml-file>

# check pods

kubectl get pods

# check pod logs

kubectl logs <pod-name>

Note: By default PODS can be accessible only with in the cluster. We can't access PODS outside the cluster directley.

=> To access PODS outside the cluster we need to expose them using K8S services concept.

===============

K8S Services

===============

=> K8S service is used to expose the pods for outside access.

=> We have 3 types of services in k8s

a) Cluster IP

b) NodePort

c) LoadBalancer

====================

What is Cluster IP

====================

=> POD is a short lived object.

=> When POD is damaged/crashed k8s will replace that with new POD.

=> For every POD new ip will be assigned.

Note: It is not at all recommended to use POD ip to access it.

=> Cluster IP service is used to group all the PODS and assign one static ip to access the pods.

Note: Using Cluster IP we can access pods only with in the cluster.

Ex: Database pods we don't expose for outside access.

===========================

What is NodePort service ?

===========================

=> NodePort service is used to expose the pods to access outside of the cluster.

=> Using NodePort we can access the pods which are running in particular worker node.

Note: Burden will be increased on particular worker node.

===============================

What is Load Balancer service

===============================

=> Load Balancer service is used to distribute the load at worker node level.

=========================

K8S Service Manifest YML

=========================

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30070

...

# check k8s services

kubectl get svc

# create service using above yml

kubectl apply -f <svc-yml>

# check services

kubectl get svc

# Open minikube tunnel for our application access

minikube service javawebappsvc

Note: If we deploy above application in EKS cluster then we should use node public ip to access our application.

URL : http://worker-public-ip:node-port/java-web-app/

===================

What is NodePort ?

==================

=> When we use service type as NodePort then k8s will use one random port number to expose our application on worker node.

Node Port Range : 30,000 - 32767

Note: If we can also fix nodeport number in service manifest yml.

========================================

POD and Service in Single Manifest YML

========================================

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javac1

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: 30070

...

===============

K8S namespaces

===============

=> Namespaces are used to group the resources logically

database-pods =====> database-ns

backend-pods ====> backend-ns

frontend-pods ===> frontend-ns

=> Inside k8s cluster we can create multiple namespace

=> Each namespace is isolated with another namespace.

# display k8s namespaces

kubectl get ns

# get pods of specific namespace

kubectl get pods -n kube-system

Note: In kubectl command if we don't specify any namespace then it consider "default" namespace.

=> In K8s we can create namespace in 2 ways

1) using "kubectl create ns" command

2) Using manifest YML file

---------------

Approach-1 :

---------------

# create namespace

kubectl create ns ashokitns

# delete namespace

kubectl delete ns ashokitns

Note: When we delete a namespace all the resources belongs to that namespace also gets deleted.

-------------

Approach-2 :

-------------

---

apiVersion: v1

kind: Namespace

metadata:

name: ashokit-backend-ns

...

# execute manifest yml

kubeclt apply -f <yml>

# get all resources belongs to ashokit namespace

kubectl get all -n ashokit

==========================

Namespace + POD + Service

==========================

---

apiVersion: v1

kind: Namespace

metadata:

name: ashokit

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

namespace: ashokit

labels:

app: javawebapp

spec:

containers:

- name: javac1

image: ashokit/javawebapp

ports:

- containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

namespace: ashokit

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30070

...

# run above yml file

kubectl apply -f <yml>

# check pods

kubectl get pods -n ashokit

# check service

kubectl get svc -n ashokit

# check all resources

kubectl get all -n ashokit

# open minikube tunnel to access app

minikube service javawebappsvc -n ashokit

=====================================

1) What is Orchestration

2) K8S introduction

3) K8S Advantages

4) K8S Architecture

5) K8S Setup (minikube & EKS)

6) What is POD

7) What is Service (ClusterIP, NodePort, LBR)

8) What is Namespace

========================================

=> When we create POD directley using "kind: Pod" in manifest yml then k8s will not manage pod life cycle.

=> If we delete pod then k8s will not create new POD.

=> If we want k8s to manage POD life cycle then we should use k8s resources to create the PODS.

1) ReplicationController (RC)

2) ReplicaSet (RS)

3) Deployment

4) DaemonSet

5) StatefulSet

==============================

What is ReplicationController

==============================

=> It is one of the resource in k8s to manage pod life cycle.

Note: If any pod is deleted/crashed/damaged then RC will perform self healing.

=> Using RC we can perform PODS count scale up and scale down

---

apiVersion: v1

kind: ReplicationController

metadata:

name: javawebrc

spec:

replicas: 2

selector:

app: javawebapp

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

...

kubectl apply -f rc.yml

kubectl get all

kubectl get pods

kubectl delete pod <pod-name>

kubectl get pods

kubectl scale rc javawebrc --replicas=5

kubectl scale rc javawebrc --replicas=1

============

ReplicaSet

============

=> It is one of the resource in k8s to manage pod life cycle.

Note: If any pod is deleted/crashed/damaged then RS will perform self healing.

=> Using RS we can perform PODS count scale up and scale down.

---

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: javawebrs

spec:

replicas: 2

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

...

kubectl apply -f rc.yml

kubectl get all

kubectl get pods

kubectl delete pod <pod-name>

kubectl get pods

kubectl scale rs javawebrs --replicas=5

kubectl scale rs javawebrs --replicas=1

============

Deployment

============

=> It is one of the resource in k8s to manage pod life cycle.

=> This is the most recommended approach to deploy our applications in k8s cluster.

=> With Deployment approach we have below advantages

1) Zero downtime

2) Auto Scaling

3) Rolling Update and Rollback

=> In Deployment we have 2 strategies to create PODS

1) RollingUpdate

2) Recreate

=> ReCreate means it will delete all existing pods and will create new pods.

Note: In This ReCreate approach, application will have downtime.

=> RollingUpdate means it will delete old pod and create new pod one after other.

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: javawebdeploy

spec:

replicas: 2

strategy:

type: RollingUpdate

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

...

=====================================================================

Assignment : Setup K8S HPA (Horizontal POD Auto Scaler)

Reference Video : https://www.youtube.com/watch?v=c-tsJrcB50I&t=667s

======================================================================

=====================

ConfigMap & Secrets

=====================

=> For every application multiple environments will be available.

1) DEV

2) SIT

3) UAT

4) PILOT

5) PROD

=> Every env will have its own config properties to run the application

Ex :

1) database props

2) smtp props

3) kafka server properties

4) redis server properties

5) payment gateways

6) third party urls

=> If we configure above properties with in the application then our application will become tightly coupled. When we want to deploy our application in another environment then we have to "change properties + re-package + re-create docker image + re-deployment". It is time taking process and risky.

=> If we want to deploy our app in multiple environments then we need to make sure our application is loosely coupled with env properties.

## To make our app loosely coupled with env properties we can use ## ConfigMap & Secrets" ##

=> Using configmap and secret we can de-couple application code and application properties so that our docker images will become loosely coupled.

Note: At the time of deployment, we can supply environment properties to the application container using ConfigMap & Secrets.

=> ConfigMap & Secret will store data in key-value format

=> ConfigMap is used to store non-sensitive data.

=> Secret is used to store sensitive data.

=========================

ConfigMap manifest YML

=========================

---

apiVersion: v1

kind: ConfigMap

metadata:

name: configmap-dev

data:

db\_url: "jdbc:mysql://localhost:3306/"

db\_name: "ashokit"

db\_port: "3306"

---

$ kubect get configmap

$ kubectl apply -f <yml>

=========================

Secret manifest YML

=========================

---

apiVersion: v1

kind: Secret

metadata:

name: secrets-dev

type: Opaque

data:

db\_username: YXNob2tpdA== #root

db\_password: YWJjQDEyMw== #abc@123

---

$ kubectl get secret

$ kubectl apply -f <yml>

===========

Assignment

===========

1) Deploy MySQL database in k8s cluster by using configmap and secret.

DB name : read from config map

DB username & pwd : read from secret