**\* \* \* \* \* 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

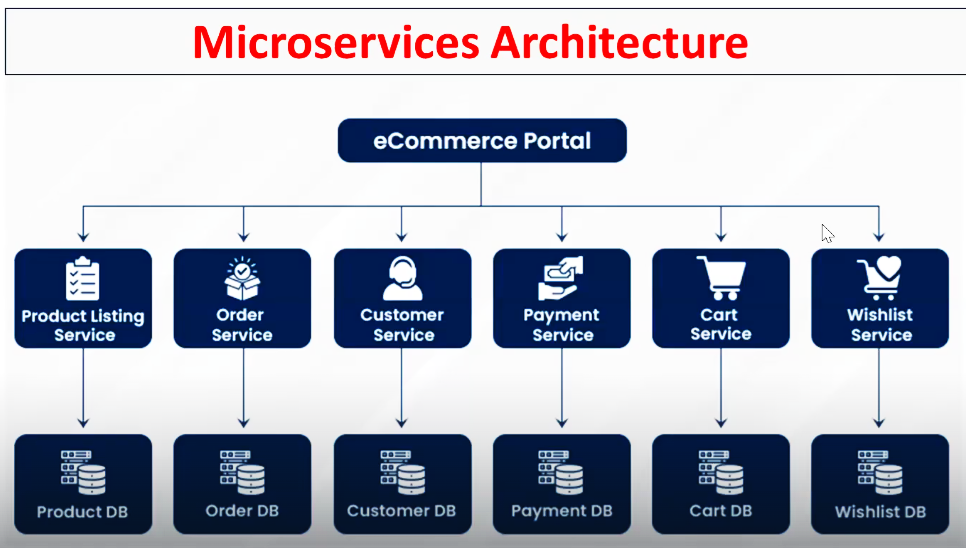
=> K8S developed by google company.

=> 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**

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

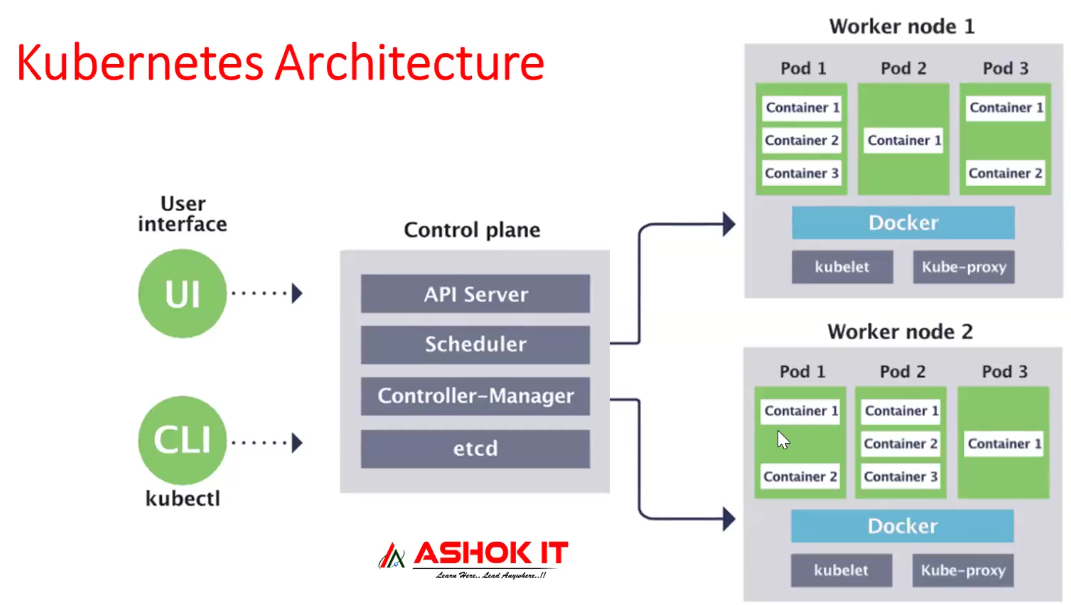
=> K8S will follow cluster architecture.

**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 tasks 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 => Single 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

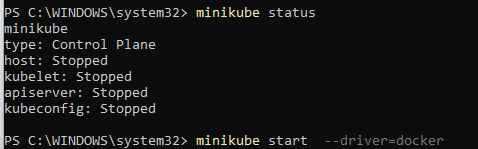
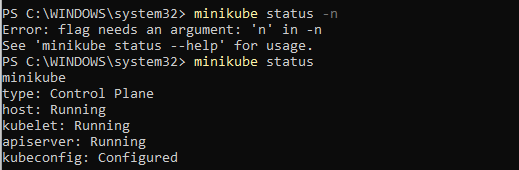
kubectl create deployment nginx-deployment --image=nginx:latest --replicas=3

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 .

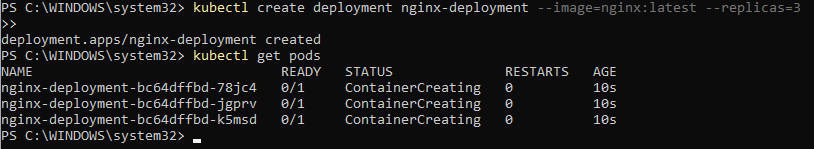
**Command – kubectl get pods**

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

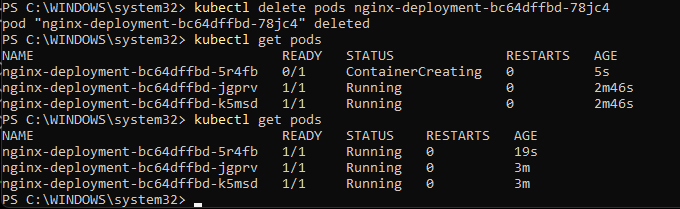
**Command – kubectl get service**

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**

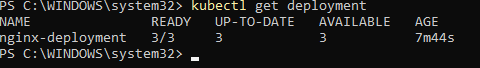


**Command – Kubectl delete pods <POD-NAME>**



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

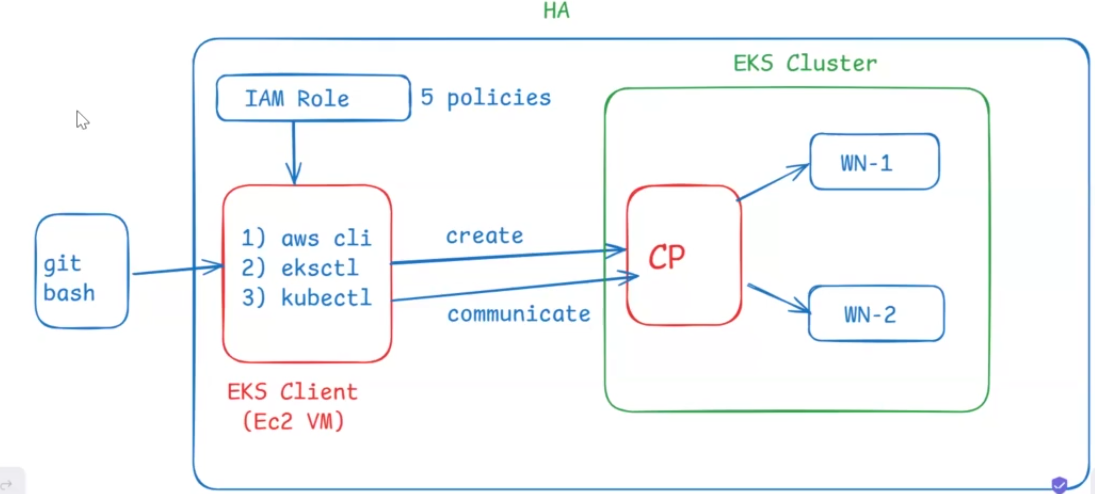
**Command – kubectl get deployment** ( deployment created by us)



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

**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 .

## **Creating EKS**

**Step 1z**

**1 :** Open Devops documents in ashokit github

2: Create EKS with Ubuntu vm using t2.micro

3: Go to the Devops-documents copy Script and paste

curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin

kubectl version --short –client

4: Install AwS Cli

* **It is a command-line utility to communicate with AWS cloud**.

sudo apt install unzip

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws –version

5: Install EKS setup

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

* **In realtime we will use terraform to create a cluster.**

Step 2

* create IAM role & attach to EKS Management Host

1. Search IAM in the AWS console
2. Click on roles in left side menu
3. Creat role 🡪 AWS service 🡪use case 🡪 EC2 🡪search permission policies
4. [AdministratorAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAdministratorAccess) 🡪[IAMFullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FIAMFullAccess) 🡪[AmazonVPCFullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonVPCFullAccess)🡪[AWSCloudFormationFullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAWSCloudFormationFullAccess)🡪

[AmazonEC2FullAccess](https://us-east-1.console.aws.amazon.com/iam/home?region=eu-north-1#/policies/details/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonEC2FullAccess) .

1. Role🡪 EKSRole

Note - Policies Can be added users directly , roles can be added to the services .

1. Go to the instace and select instance 🡪action 🡪security🡪edit IAM Role🡪select role🡪update

# Step 3 : Create EKS Cluster using eksctl

# Copy N verginia to the notepad

eksctl create cluster --name ashokit-cluster4 --region eu-north-1 --node-type t2.medium --zones eu-north-1a,eu-north-1b

1. Go to the gitbash and execute the above command .

eksctl create cluster --name ashokit-cluster4 --region us-east-2 --node-type t2.medium --zones us-east-2a,us-east-2b

* Check region as per your region

1. Check cluster is created or not

Command – kubectl get nodes

Command – kubectl get pods

Command – kubectl cluster-info

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

Command – kubectl get pods

1. delete one pod

Command – kubectl delete pod <POD-NAME>

1. Command – Kubectl delete all –all It will delete all the pods permantely .
2. We need to deploy one java web application
3. Got to the github <https://github.com/ashokitschool/kubernetes_manifest_yml_files>
4. Open [01-RC-RS-Deployment](https://github.com/ashokitschool/kubernetes_manifest_yml_files/tree/main/01-RC-RS-Deployment) 🡪 [02\_java\_web\_app\_lbr.yml](https://github.com/ashokitschool/kubernetes_manifest_yml_files/blob/main/01-RC-RS-Deployment/02_java_web_app_lbr.yml) copy file and open gitbash create new file deploy.yml and paste that code into the file and save.
5. To run deploy.yml file run this command 🡪 kubectl apply –f deploy.yml

Note – after practice completion delete all the cluster otherwise it will generate bill.

**Command - eksctl delete cluster --name ashokit-cluster4 --region ap-south-1**

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

**What is POD ? 12-28-k8s-28-Nov-24**

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

=> POD is a smallest building block it is used to run our application in the k8s cluster.

=> Applications will be deployed as PODS in k8s.

**Note: To create PODS we will use docker images. Without docker image we can’t create a POD.**

=> 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).

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

**Nodes**

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

* Nodes are the worker machines in the cluster where Pods are scheduled to run.
* Each Node runs a container runtime (e.g., Docker, containerd), the kubelet (to manage the Node), and other services needed for Kubernetes.

**Types of Nodes:**

* **Control Plane Nodes (Master Nodes):** Manage the cluster and handle scheduling, API requests, and overall orchestration.
* **Worker Nodes:** Execute application workloads (Pods).

**Key Features:**

* Nodes have a fixed amount of CPU, memory, and storage resources that they make available for Pods.
* Nodes communicate with the control plane to report health and resource usage.
* The scheduler determines which Pods run on which Nodes based on available resources and policies.
* **Management:** Use kubectl to interact with Nodes (e.g., kubectl get nodes, kubectl describe node <node-name>).

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

**K8S manifest YML**

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

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

**apiVersion: <version-number>**

* Vesion number of the resource we are using .

**kind: <resource-type>**

* What we want to create service , POD we specify here .

**metadata: <name>**

* Name of the resource .

**spec: <container>**

* Specify container information in the POD

**Label :** is used to identify running PODS in the cluster .

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

**POD MANIFEST YAML**

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

--- ( Three hyphen represent starting point )

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javac1 ( Here dash represent array value, array means group of values )

image: ashokit/javawebapp

It will fetch image from docker hub

ports:

- containerPort: 8080 (Container port is a application port number )

… ( Three dot represent ending point )

* Open gitbash and check minikube status
* If container are stopped the start the container

Command – minikube start --driver=docker

* Then create on yml file ex.- vi 01-pod.yml and copy above script and paste
* Check pod is created or not command – kubectl get pods

**Note: Save above content in .yml file**

# execute manifest yml

kubectl apply -f <file-name>

# check pods

kubectl get pods

# check pod logs

kubectl logs <pod-name>

**Note:** By default POD’S can be accessible only with-in the cluster. We can't access PODS outside the cluster directly.

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

* In the 01-pod.yml file we are giving yml file as a input , kubernetes is going to read this yml , kubernetes will understand configuration of yml .
* In the yml ,container section I am asking kubernetes to create a container by using my docker image , I’m asking kubernetes to create a pod .
* Kubernetes will search for the docker image in the YAML file and if the image is available then it will download the image , With the docker image it will create a POD.
* Label is important in the POD.
* Kubectl is used to communicate with the cluster and it is used to perform the opreations.
* Creating a POD means we are deploying our application in the kubernetes.
* We cant access a POD outside the cluster.
* We should not depend on
* Cluster ip is used to group all the pods to one static-ip and we can access within the cluster ,database pods we can access using cluster ip .
* Load should be distributed at the worker node level as well as the pod level .

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

* **Cluster** means only inside cluster pods we can access .
* **Node-port** means we can access pod outside cluster but only one machine pods.
* **Load-balancer** means all machine pods of all worker nods we can access.

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

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

**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 Node-Port service ? 13-28-k8s-29-Nov-24 ===========================**

=> Node-Port service is used to expose the pods to access outside of the cluster.

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

Note: Burden will be increased on particular worker node.

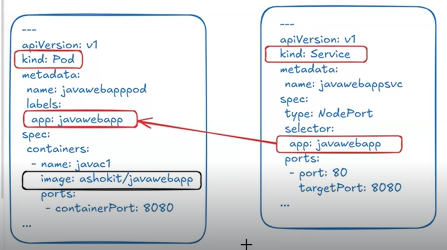
**===============================**

**What is Load Balancer service**

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

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

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

****

**First YAML file will create the POD and the Second YAML will expose the POD.**

**What is Label and service selector in kubernetes ?**

**Label :** lable is used to identify the pods inside the service we are going to configure the label as a selector so the service is going to identify the POD.

Selector is going to identify using pod labels , we are going to identify pod using selector.

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

**K8S Service Manifest YML**

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

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

nodePort: 30070

...

#Create 01-pod.yml and 02-service.yml file and the apply command

Kubectl apply –f 01-pod.yml

#check pods

Kubectl get pods

# 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 ( service-name)

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

Which service is deployed on which worker node that is decided by kubernetes control-plane.

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

$ minikube service javawebappsvc

|-----------|---------------|-------------|---------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|---------------|-------------|---------------------------|

| default | javawebappsvc | 80 | http://192.168.49.2:30070 |

|-----------|---------------|-------------|---------------------------|

\* Starting tunnel for service javawebappsvc.

|-----------|---------------|-------------|------------------------|

| NAMESPACE | NAME | TARGET PORT | URL |

|-----------|---------------|-------------|------------------------|

| default | javawebappsvc | | http://127.0.0.1:65175 |

|-----------|---------------|-------------|------------------------|

Copy second ip address | <http://127.0.0.1:65175/java-web-app/>

And run in the browser , you will now able to access application.

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

**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 # Service-port

targetPort: 8080 (Application port)

nodePort: 30070 (Machine port) Only node port we need to enable in inbound rulesEKS

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

**K8S namespaces 14-28-k8s-02-DEC-24**

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

=> 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 is the default namespace in 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

...

When you are going to create a container inside the pod , Then only specification only required.

But when we want to create a namespace then specification not required.

# execute manifest yml

**Command - kubeclt apply -f <yml> For every file command is comman.**

# get all resources belongs to particular ashokit namespace

**Command - 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 directly 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 Replication-Controller will perform self healing.

=> Using Replication-Controller we can perform PODS count scale up and scale down.

**Note - RC will support only one selector . we are going to identify pod using selector when we go for RC.**

---

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

Creating a pod manually is not recommended .

Creating a pod using resources is recommended . the benefit is that k8s will take care of our pod life-cycle.

* We can scale up and scale-down pods depends upon the requirement.

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

**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.

**Note - RS will support multiple selectors by using matchLabels we can configure**

---

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**

* If we have done wrong deployment then immediately we can switch to the old application.

=> 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

=====================================================================Github Link : - <https://github.com/ashokitschool/k8s_metrics_server>

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

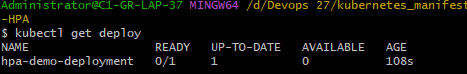


Step 1: Install Metrics server

How to install metrics server in the minikube

* Clone above repository to the target folder
* Open git bash and go 🡪 /d/Devops 27/k8s\_metrics\_server/deploy/1.8+
* Run command – kubectl apply –f . our metrics server will get installed .
* Check metrics server installed or not kubectl top nodes if not then run command
* kubectl get pods -n kube-system | grep metrics-server
* 

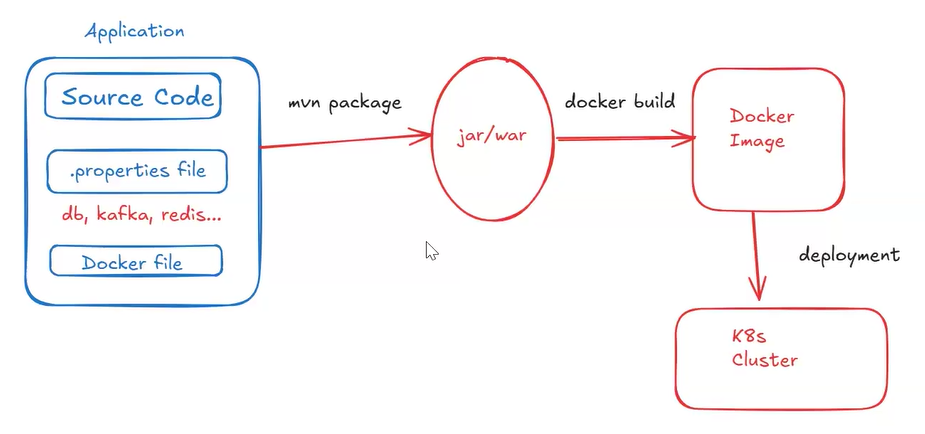
Step 2 : Deploy Sample App

* Clone repo <https://github.com/ashokitschool/kubernetes_manifest_yml_files.git>
* Go to the /d/Devops 27/kubernetes\_manifest\_yml\_files/05-HPA
* Run command kubectl apply –f . & kubectl get deploy
* 
* Lets expose the service by using 02-service.yml file
* kubectl apply -f 02\_Service.yml & kubectl get 02\_Service.yml
* check kubectl get svc
* 
* Now We need to Create HPA
* 03\_HPA.yml file is available
* Kubectl get hpa

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

**ConfigMap & Secrets 16-28-k8s-05-DEC-24**

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



* In (. Properties) file database details , kafka details , redis details .
* If we have modified the ( .properties ) file the we need to **re-package + re-create docker image + re-deployment .**

=> 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 properties

2) smtp properties

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**

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

/d/Devops 27/K8S

---

**apiVersion:** v1

**kind:** ConfigMap

**metadata:**

**name:** configmap-dev

**data:**

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

**db\_name:** "ashokit"

**db\_port:** "3306"

---

$ kubectl get configmap

$ kubectl apply -f <yml>

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

**Secret manifest YML**

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

---

apiVersion: v1

kind: Secret

metadata:

name: secrets-dev

Opaque is a encryption type

type: Opaque

data:

db\_username: YXNob2tpdA== #root

Encrypted Format

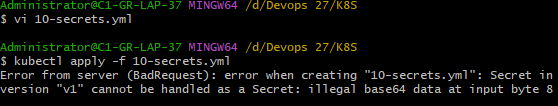
db\_password: YWJjQDEyMw== #abc@123

---

$ kubectl get secret

$ kubectl apply -f <yml>

Error



Solution

$ echo -n "ashokit" | base64

YXNob2tpdA==

$ echo -n "abc@123" | base64

YWJjQDEyMw==

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

**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

**17-28-k8s-06-DEC-24**

1) create configmap.yml (ex: mysql-configmap.yml)

- is used to maintain database details , db-name and db-username.

- it is used to store the data required for the database setup.

2) create secret.yml (ex: mysql-secret.yml)

- We can maintain password in the secret

3) create deployment.yml (ex: mysql-deployment.yml)

- deployment is used for deploying the pods.

4) service.ymll is optional

configmap.yml

* Will configure database name and user in cofigmap.yml

---

apiVersion: v1

kind: ConfigMap

metadata:

name: mysql-config-map

data:

MYSQL\_DATABASE: mydatabase

MYSQL\_USER: myuser

...

---

apiVersion: v1

kind: Secret

metadata:

name: mysql-secret

type: Opaque

data:

MYSQL\_ROOT\_PASSWORD: cm9vdA==

MYSQL\_PASSWORD: cm9vdA==

....

POD

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: mysql-deployment

spec:

replicas: 1

strategy:

type: ReCreate

selector:

matchLabels:

app: mysql

template:

metadata:

labels:

app: mysql

spec:

containers:

- name: mysql-container

image: mysql:latest

ports:

- containerPort: 3306

How to read the the data from configmap & secret , how to give data to the mysql container ?

Que- How to pass environment properties to the container?

* If I want pass properties we need to use **env** keyword , we need to set env variable

env:

- name: MYSQL\_DATABASE

Predefined env properties

valueFrom:

From the configmap

configMapKeyRef:

name: mysql-config-map

key: MYSQL\_DATABASE

- Kubernetes will identify configmap with the name and in the config-map it will identify the key , it will get the value of the key and store it into the MYSQL\_DATABASE.

- name: MYSQL\_USER

valueFrom:

configMapKeyRef:

name: mysql-config-map

key: MYSQL\_USER

* User is also available in the configmap.yml ,identify the key and get the value of the key from the configmap.yml

- name: MYSQL\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-secret

key: MYSQL\_ROOT\_PASSWORD

* MYSQL\_ROOT\_PASSWORD is available in the mysql-secret , we need to get the value from the key available in the mysql- secret resource.

**Note - This is only for the root users**

- name: MYSQL\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-secret

key: MYSQL\_PASSWORD

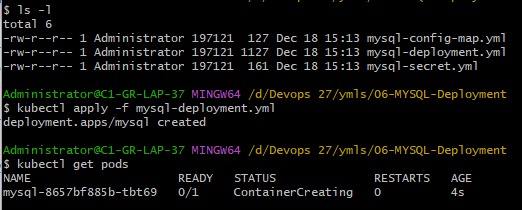
...

* This is for normal users
* Properties names are always in Uppercase , because it is case sensitive .

Steps: -

#Check pods

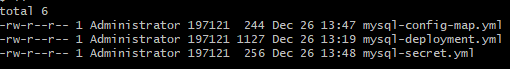
Kubectl get pods



* Container is created and mysql database is configured.

**Error -**If you get ContainerCreatingError Then

Run all the files and then check.



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

**Blue - Green Deployment Approach 17-28-k8s-06-DEC-24**

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

**It is used to deploy application to the production.**

=> Blue green deployment is an application release model

=> It reduces risk and minimizes downtime

=> It uses two production environments, known as Blue and Green

=> The old version can be called the blue environment

=> The new version can be known as the green environment

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

**Advantages**

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

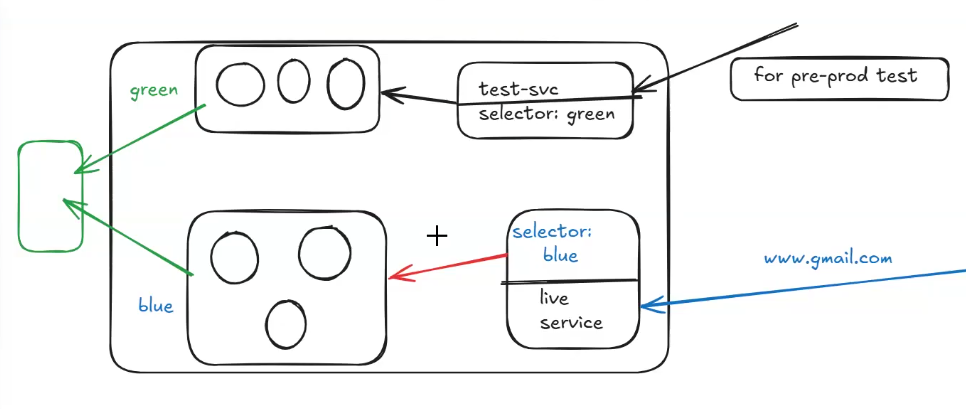
=> Rapid releasing

=> Simple rollbacks

=> Built-in disaster recovery

=> Seamless customer experience

=> Zero Downtime



* First will create a pod and give selector name as blue
* Live service is mapped to the domain name 🡪 customer access this domain name that request goes to the 🡪 live service .
* live service identifying pods by using selector as blue , blue pods are serving live traffic.
* Live traffic is going to live service live service exposed blue pods .

Now we want to release latest code to the production , Then we are going to create latest pods in the kubernetes cluster with the latest docker image .

I’m going to give name to the latest pods as green.

For application three pods are running with the blue lable and three pods are running with the green lable.

Difference between in green lable and blue lable is

* Green pods are created with the latest docker image .
* Blue pods are created with the old docker image.

Latest code is available in the green pods but the traffic is going to the blue pods , because we have given selector as a blue . so it wil forward request to the blue pods , in the blue pods old code is running.

We are going to create one more service and wil keep name as green test service and the traffic coming to the green test service is expose pods with the lable as a green.

If testing is completed then we need to divert the live traffic to the latest deployment.

* For that we need to change the lable from blue to the green and re-apply the service.
* After applying service the traffic is diverted to the green .

If the latest pods is not working as expected then will change selector green to the blue. So the traffic will diverted to the old blue pods.

* First we have done the application deployment and given name as blue .
* Then we created a live service , live service identify by using selector as blue.
* Then live service name is mapped to the domain name.
* Whenever request comes to the domain , domain will forward request to the live service.
* Then we have updated the code with some updated features for that we have created the green pods and to test pods we’ve have created a test service with the selector as a green. And the latest code is working expected then will deploy latest code to the production.
* For that change lable as green and re-apply service file.

## Step-1 : Create blue deployment (pods will be created)

## Step-2 : Check pods status

## Step-3 : Create Live service to expose green pods

pods (Type : Load Balancer)

Ex : Selector is v1

## Step-4 : Access Load Balancer URL

Ex: http://lbr-public-dns/java-web-app/

## Step-5 : Create Green Deployment (pods will be created with latest docker image and label as v2).

## Step-6 : Verify green pods status

## Step-7 : Make green pods as live by changing "live-service" selector as 'v2' in yml file. After changing yml then re-execute it.

## Step-8 : Access load balancer URL and see the response of application.

Ex: http://lbr-public-dns/java-web-app/

Note: If required we can switch from green pods to blue pods by changing live-service yml selector from v2 to v1.

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

What is DaemonSet ?

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

=> It is one of the resource in k8s which is used to manage PODS life cycle.

=> It is used to create copy of the pod in every worker node.

Ex: FluentD pods will be created using DaemonSet.

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

What is StatefulSet ?

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

=> It is one of the resource in k8s cluster which is used create stateful pods.

Ex : ElasticSearch pods will be created using StatefulSet.

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

Application Log Monitoring

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

=> In Real-time application will be deployed with multiple pods for high availability.

=> Application pods will run in multiple worker nodes.

Note: As multiple pods running in multiple worker nodes for application multiple log files will be created.

Note: If any problem occurs in the application then to find root cause of the problem we need to check all log files of the application which is difficult task.

=> To overcome this problem we will use Log centralization concept.

=> To centralize log monitoring of the application we will use ELK/EFK stack setup.

ELK : Elastic Search + Log Stash + Kibana

EFK : Elastic Search + FluentD + Kibana

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

Assignment : EFK Stack setup in K8S Cluster

@@ Reference video : https://youtu.be/8MLcbbfEL1U?si=bQ\_BrOv3EiLu48eu

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

What is HELM ?

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

=> In linux we will use package managers to install a software

Ex : yum , apt, rpm etc...

=> HELM is a package manager which is used to install required softwares in k8s cluster.

=> HELM will use charts to install required packages.

=> Chart means collection of configuration files (manifest ymls).

=> Using HELM chart we can install promethues server

=> Using HELM chart we can install grafana server

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

Helm Installation

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

$ curl -fsSl -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3

$ chmod 700 get\_helm.sh

$ ./get\_helm.sh

$ helm

-> check do we have metrics server on the cluster

$ kubectl top pods

$ kubectl top nodes

# check helm repos

$ helm repo ls

# Add the metrics-server repo to helm

$ helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/

# Install the chart

$ helm upgrade --install metrics-server metrics-server/metrics-server

$ kubectl top pods

$ kubectl top nodes

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

Kubernetes Monitoring

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

=> We can monitor our k8s cluster and cluster components using below softwares

1) Prometheus

2) Grafana

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

Prometheus

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

-> Prometheus is an open-source systems monitoring and alerting toolkit.

-> Prometheus collects and stores its metrics as time series data

-> It provides out-of-the-box monitoring capabilities for the k8s container orchestration platform.

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

Grafana

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

-> Grafana is an analysis and monitoring tool.

-> It provides visulization for monitoring.

-> It provides charts, graphs, and alerts for the web when connected to supported data sources.

Note: Graphana will connect with Prometheus for data source.

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

How to deploy Grafana & Prometheus in K8S

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

-> Using HELM charts we can easily deploy Prometheus and Grafana

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

Install Prometheus & Grafana In K8S Cluster using HELM

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

# Add the latest helm repository in Kubernetes

$ helm repo add stable https://charts.helm.sh/stable

# Add prometheus repo to helm

$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

# Update Helm Repo

$ helm repo update

# install prometheus

$ helm install stable prometheus-community/kube-prometheus-stack

# Get all pods

$ kubectl get pods

Node: You should see prometheus pods running

# Check the services

$ kubectl get svc

# By default prometheus and grafana services are available within the cluster as ClusterIP, to access them outside lets change it to LoadBalancer.

# Edit Prometheus Service & change service type to LoadBalancer then save and close that file

$ kubectl edit svc stable-kube-prometheus-sta-prometheus

# Now edit the grafana service & change service type to LoadBalancer then save and close that file

$ kubectl edit svc stable-grafana

# Verify the service if changed to LoadBalancer

$ kubectl get svc

=> Access Promethues server using below URL

URL : http://LBR-DNS:9090/

=> Access Grafana server using below URL

URL : http://LBR-DNS/

=> Use below credentials to login into grafana server

UserName: admin

Password: prom-operator

=> Once we login into Grafana then we can monitor our k8s cluster. Grafana will provide all the data in charts format.

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

Node Selector

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

=> Node Selector is used to schedule the pods on particular worker node only.

=> To achieve this we can assign label for the worker node and we can configure that node label in our manifest yml as node selector.

Note: If node selector is matching with worker node label then our pods will be created on that particular worker node. If node-selector not matching with worker node label then pods will not be scheduled for execution.

=> Execute below manifest yml to create nginx deployment with 3 pod replicas

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

nodeSelector:

name: ashokit-wn-1

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

...

$ kubectl apply -f <yml>

$ kubectl get pods

Note: here pods will be in pending state beause no worker node label is matching with node selector configured in manifest yml file.

# configure label for worker node

$ kubectl get nodes

$ kubectl edit node <node-name>

# configure below lable under labels section

name: ashokit-wn-1

$ kubectl get pods

Note: here pods will come into running state.

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

Node Affinity

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

=> Node Affinity preffered approach. If nodeSelector is matching with any worker node label then schedule pods on that worker node only.

=> If matching is not found then schedule pods on any available worker node in the cluster.

---

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:

affinity:

nodeAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 1

preference:

matchExpressions:

- key: name

operator: In

values:

- ashokit

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

...

$ kubectl apply -f <yml>

$ kubectl get pods -o wide

========

Taints

========

=> Taints are used to make worker node not eligible for pods scheduling.

=> We have 3 popular taints

1) No Schedule

2) No Execute

3) Prefer No Scedule

# create taint on worker node

$ kubectl taint nodes <node-name> color=blue:NoSchedule

# remove taint from worker node

$ kubectl taint nodes <node-name> key1=value1:NoSchedule-

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

Tolerations

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

=> Tolerations are used to schedule pods on tainted worker nodes also.

# tainting worker node

$ kubectl taint nodes <node-name> key1=value1:NoSchedule

Note: If any pod is having tolerations as key1 and value1 then schedule those pods even though the nodes are tainted with NoSchedule state.

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

tolerations:

- key: "key1"

operator: "Equal"

value: "value1"

effect: "NoSchedule"

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

...

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

9) ReplicationController (RC)

10) ReplicaSet (RS)

11) Deployment

12) DaemonSet

13) StatefulSet

14) ConfigMap and Secrets

15) Blue-Green Deployment

16) EFK Stack

17) HELM Charts

18) K8S Monitoring

19) Node Selector

20) Node Affinity

21) Taints & Tolerations