Git Repo : https://github.com/ashokitschool/kubernetes_manifest_yml_files.git

==========
Kubernetes (K8S)
==========

=> It is free & open source s/w.

=> Google developed this k8s using "GO" programming language.

=> K8S provides Orchestration (Management) platform.

=> K8S is used to manage containers
(create/stop/start/delete/scale-up/scale-down)

==========
Advantages
==========

1) Container Orchestration : Manages containers.

2) Self Healing : If any pod got crashed/deleted then it will replace with new pod.

3) Load Balancing : Load will be distributed all containers which are up and running.

4) Auto Scaling : Based on demand containers count will be increased or decreased.

==========
Docker Vs Kubernetes
==========

Docker : Containerization platform

Note: Packaging our app code and dependencies as single unit for execution is called as Containerization.

Kubernetes : Orchestration platform

Note: Orchestation means managing the containers.

==========
Kubernetes Architecture
==========

=> K8S will follow cluster architecture.

=> Cluster means group of servers will be available.

=> In K8s Cluster we will have master node (control plane) and worker nodes.

==========
K8S Cluster Components
==========

1) Control Node (Master Node)

- API Server
- Schedular
- Controller Manager
- ETCD

2) Worker Nodes

- Kubelet
- Kube Proxy
- Docker Runtime
- 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 by using 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" will verify all the taks are working as expected or not.

=> In Worker Node, "Docker Engine" will be available to run docker container.

=> 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 Cluster 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

Note: Provider will give ready made cluster for us.

Note: Provider Managed Clusters are chargable

==========
Assignment : Setup MiniKube software
==========

Linux VM : https://github.com/ashokitschool/DevOps-Documents/blob/main/13_MiniKube_Setup.md

Windows : Download install docker desktop + download and install minkube

https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2F.exe+download

==========
ðŸ‘‰ *AWS EKS Setup Explained By Mr. Ashok*: https://www.youtube.com/watch?v=is99tq4Zwsc

ðŸ‘‰ *AWS EKS Setup Doc* : https://github.com/ashokitschool/DevOps-Documents/blob/main/05-EKS-Setup.md
==========

==========
What is POD ?
==========

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

=> Our application will be deployed as a POD only in k8s.

=> To create PODS we will use Docker images.

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

=> If we run our application with multiple PODS then we will get high availability.

=> Based on our demand we can increase our PODS count and we can decrease our PODS count.

=> To create PODS in k8s we will use manifest YML file.

==========
What is YML?
==========

=> YML stands for YET another markup language.

=> It is used to store the data in key-value format.

=> YML files are both human and machine readable.

=> YML file we can save with .yml or .yaml extension.

=> Official website : www.yaml.org

==========
01-Sample YML FILE
==========

```
---
id: 1234
name: John
gender: Male
skills:
 - Linux
 - AWS
 - DEVOPS
...
```

==========
02-Sample YML FILE
==========

```
employee:
```

```
 id: 68686
 name: Smith
 address:
  city: Hyd
  state: TG
  country: India
```

```
==================
K8S Manifest YML
==================
```

=> To deploy our application in kubernetes we need MANIFEST YML file.

=> In k8s manifest YML we will write 4 sections.

```
apiVersion: <resource-version-number>

kind: <resource-type>

metadata: <resource-info>

spec: <container-info>
```

```
==================
POD Maniest YML
==================
```

```
---
apiVersion: v1
kind: Pod
metadata:
 name: javawebapppod
 labels:
  app: javawebapp
spec:
 containers:
  - name: javac1
    image: ashokit/javawebapp
    ports:
     - containerPort: 8080
...
```

Note: Save the above content in .yml file.

```
$ kubectl get pods

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

$ kubectl get pods

$ kubectl logs <pod-name>

# display in which worker node our pod is running
$ kubectl get pods -o wide
```

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

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

```
===============
K8S Service
===============
```

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

=> In K8S we have 3 types of services

1) Cluster IP
2) Node PORT
3) Load Balancer

=> When we deploy our application in k8s then PODS will be created for our application..

=> For every POD one IP will be generated.

=> PODS are short lived objects. We should not access PODS using POD IP because PODS can be destroyed and re-created at any point of time.

CLUSTER IP : It generates one static IP for all PODS based on POD label.

Ex: 192.168.10.89

Note: using ClusterIP we can access PODS only with in the cluster.

Note: PODS created , PODs Destroyed, PODS increased or decreased still ClusterIP will not change.

Ex: Database PODS we should access only within in the cluster. Outside ppl should not access DB pods. In this scenario we can use CLUSTER IP service.

Node PORT : It is used to expose the PODS for outside access.

Using NODE PUBLIC IP we can access PODS running in that node outside of the cluster also.

LOAD BALANCER : It is used to expose the PODS for outside access.

=> When we can access LOAD BALANCER URL, it will distribute the load to all the nodes and all the PODS available for our application.

==========
K8S Service Manifest YML
==========

```
---
apiVersion: v1
kind: Service
metadata:
 name: javawebappsvc
spec:
 type: NodePort
 selector:
  app: javawebapp
 ports:
  - port: 80
    targetPort: 8080
    nodePort: 30070
...
```

Note: Save this data with .yml extesion

```
$ kubectl get svc

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

$ kubectl get svc
```

Note: Enable NODE PORT in security group inbound rules.

Note: To access our application use below URL

URL : http://worker-node-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 for public access.

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
...
```

```
$ kubectl apply -f <manifest-yml-name>

$ kubectl get pods

$ kubectl get svc
```

==========
K8S namespaces
==========

=> Namespaces are used to group the resources logically.

mysql-db-pods =====> mysql-db-ns

backend-app-pods =====> backend-ns

frontend-app-pods =====> frontend-ns

```
=> Inside k8s cluster we can create multiple namespaces.

=> Each namespace is isolated with another namespace.

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


# 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 will 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
...

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


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

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

  => If we delete pod then k8s will not create new POD in this scenario. Self Healing will not work in
  this way.

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

                                          1) ReplicationController (RC) (outdated)

                                          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:
   labels:
    app: javawebapp
  spec:
   containers:
    - name: javawebct
      image: ashokit/javawebapp
      ports:
      - containerPort: 8080
...

$ kubectl apply -f rc.yml

$ 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.

=> In Deployment we have 2 strategies to create PODS

                                1) Rolling Update

                                2) ReCreate

=> ReCreate means it will delete all existing pods and will create new pods. In This ReCreate
approach, application will have downtime.

=> RollingUpdate means it will delete old pod and create new pod one after other. Here no downtime
for our application.

---
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
...

kubectl apply -f rc.yml

kubectl get all

kubectl get pods
```

```
kubectl delete pod <pod-name>

kubectl get pods

kubectl scale deployment javawebdeploy --replicas=5

kubectl scale deployment javawebdeploy --replicas=1


==============================
HPA (Horizontal POD Scalar)
==============================

=> HPA is used to scale up and scale down our pods automatically based on traffic.

=> HPA will not work directley, we need to configure HPA in k8s cluster.

@@@ Reference Video : https://www.youtube.com/watch?v=c-tsJrcB50I

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


====================
What is DaemonSet ?
====================

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

=> Automatically adds the pod to new nodes as they join the cluster.

=> Removes the pod from nodes when they are removed.

=> Ensures one pod per node (unless explicitly configured otherwise).


#### Use Cases: Running cluster-wide background services like:

=> Log collection agents (e.g., Fluentd, Filebeat, LogStash)

=> Monitoring agents (e.g., Prometheus Node Exporter)

=> Metrics Server


# Create fluentd pods using daemonset
$ kubectl apply -f https://k8s.io/examples/controllers/daemonset.yaml

# check fluentd pods
$ kubectl get pods -n kube-system

# delete fluentd pods by deleting deamonset we have created
$ kubectl delete daemonset fluentd-elasticsearch -n kube-system


======================
what is statefulset
======================

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

=> StatefulSet is used to manage stateful application releated pods.

      stateless pod = when POD deleted POD data also gets deleted.
```

statefull pod = when pod delete POD data will not be deleted.

Note: To make our POD as stateful we need to use Storage for the POD.

ðŸ”‘ Key Features of StatefulSet:

Stable Pod Names : Pods are created with persistent names like myapp-0, myapp-1, etc. Even if a pod is deleted, its name remains the same when it is recreated.

Stable Storage : Each pod gets a persistent volume that is tied to it. The volume remains even if the pod is deleted and re-created.

ðŸ§ Use Cases:

1) Databases (MySQL, PostgreSQL, MongoDB)

2) Distributed systems (Kafka, Cassandra, Elasticsearch)

==========
Blue - Green Deployment Approach
==========

=> Blue green deployment is an application release model to the production.

=> 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 (v1)

=> The new version can be known as the green environment (v2)

==========
Advantages
==========

=> Rapid releasing

=> Simple rollbacks

=> Seamless customer experience

=> Zero Downtime

Step-1 : Create blue deployment (pods will be created with label as v1)

Step-2 : Check pods status

Step-3 : Create Live Service to expose blue pods (Type : load balancer)

Step-4 : Use Load Balancer URL To acess our app which is running in blue pods.

URL : http://lbr-url/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 live-service yml file.

Step-8 : Use Load Balancer URL To acess our app which is running in green pods.

URL : http://lbr-url/java-web-app/

==========
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 apis 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.

Note: We need to keep environment properties outside of the application code.

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: ashokit-cg-dev
data:
 db_url: "jdbc:mysql://localhost:3306/"
 db_name: "ashokit"
 db_port: "3306"
...

$ kubect get configmap
$ kubectl apply -f <yml>
$ kubect get configmap

========================
Secret manifest YML
========================

---
apiVersion: v1
kind: Secret
metadata:
 name: ashokit-secret-dev
type: Opaque
data:
 db_username: YXNob2tpdA== #root
 db_password: YWJjQDEyMw== #abc@123
...

$ kubectl get secret

$ kubectl apply -f <yml>

$ kubectl get secret

=============================================
how to read the data from configmap and secret
=============================================

env:
 - name: MYSQL_DATABASE
   valueFrom:
    configMapKeyRef:
     name: ashokit-cg-uat
     key: db_name


env:
 - name: MYSQL_PASSWORD
   valueFrom:
    secretMapKeyRef:
     name: ashokit-secret-dev
     key: db_password


===========
Assignment
===========

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

```
  DB username : read from config map
  DB pwd  : read from secret


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

---
apiVersion: v1
kind: ConfigMap
metadata:
 name: mysql-config-map
data:
 MYSQL_DATABASE: mydatabase
 MYSQL_USER: myuser
...

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

---
apiVersion: v1
kind: Secret
metadata:
 name: mysql-secret
type: Opaque
data:
 MYSQL_ROOT_PASSWORD: cm9vdA==
 MYSQL_PASSWORD: cm9vdA==
...

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

---
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

      env:
       - name: MYSQL_DATABASE
         valueFrom:
          configMapKeyRef:
           name: mysql-config-map
           key: MYSQL_DATABASE

       - name: MYSQL_USER
         valueFrom:
          configMapKeyRef:
```

```
            name: mysql-config-map
            key: MYSQL_USER

        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
           secretKeyRef:
            name: mysql-secret
            key: MYSQL_ROOT_PASSWORD

        - name: MYSQL_PASSWORD
          valueFrom:
           secretKeyRef:
            name: mysql-secret
            key: MYSQL_PASSWORD
...

# To get pods
$ kubectl get pods

# Go enter into POD
$ kubectl exec -it <pod-name> -- /bin/bash

# Connect with mysql database
$ mysql -u root -proot

# display databases available
$ show databases;

# select database
$ use mydatabase;

# show tables
$ show tables;

# exit from the mysql
$ exit
$ exit

================
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.

    Ex: Metrics Server, Promethues, Grafana, ELK stack....

=> HELM will use charts to install required packages.

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

==================
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 version

========================
```

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 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.

Note: Using HELM charts we can easily deploy Prometheus and Grafana in K8S Cluster

==========
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 & grafana
$ helm install stable prometheus-community/kube-prometheus-stack

# Get all pods
$ kubectl get pods
```

Node: You should see prometheus pods running

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

```
            $ kubectl edit svc <service-name>
```

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

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

```
=> Access Promethues server using below URL

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


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

$ kubectl edit svc stable-grafana

=> Access Grafana server using below URL

    URL : http://LBR-DNS/


=> Use below credentials to login into grafana server

UserName: admin
Password: prom-operator

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

Assignment : EFK stack setup in k8s cluster to monitor application logs

@@ Reference video : https://youtu.be/8MLcbbfEL1U?si=bQ_BrOv3EiLu48eu

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


===============
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 will configure that node label in
our manifest yml as node-selector.

    # configure label for worker node
    $ kubectl get nodes
    $ kubectl edit node <node-name>

    # configure below lable under labels section
    name: ashokit-wn-1

Note-1: If node-selector is matching with worker-node label then our pods will be created on that
particular worker-node only.

Note-2: 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 -o wide

  ===============
  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
               - ait
        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 and pods execution.

=> We have 3 popular taint options

1) No Schedule : Kubernetes will not schedule new pods on the node.

2) No Execute : New PODS will not be scheduled and Existing pods also will be removed.

3) Prefer No schedule : Kubernetes tries to avoid scheduling pods on the node but will schedule them
if no better options exist.

# create taint on worker node
$ kubectl taint nodes <node-name> color=blue:NoSchedule

# remove taint
$ kubectl taint nodes <node-name> color-


============
Tolerations
============

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

    Taint: "I donâ€™t want pods on me unless they tolerate me."

    Toleration: "Iâ€™m okay to run on a node with this taint."

# 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
...

==========================
Liveness & Readines Probes
==========================

Probes are checks performed by the Kubernetes kubelet to monitor the health of your application
running inside a pod.


## Liveness Probe :

Purpose: Indicates whether the pod is alive or dead.

If it fails: Kubernetes kills the container and may restart it based on the restart policy.

Used to detect: Crashes, deadlocks, or stuck apps.

## Readiness Probe :

Purpose: Indicates whether the pod is ready to serve traffic.

If it fails: The pod is removed from Service endpoints, so it won't receive requests.

Used during: App startup and while running.

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


1) What is Orchestration and Why

2) K8S introduction

3) K8S Advantages

4) Kubernetes Architecture

5) K8S Setup (AWS EKS)

6) PODS

7) Services (ClusterIP, NodePort & LoadBalancer)

8) Namespaces

9) ReplicationController

10) ReplicaSet

11) Deployment

12) HPA

13) Blue Green Deployment

14) ConfigMap & Secret

15) MySQL Deployment in k8s cluster

16) HELM Charts

17) Promethues & Grafana
```

18) EFK Stack

19) What is Node Selector

20) What is Node Affinity

21) What are Taints & Tolerations

22) Liveness & Readiness Probes