**Kubernetes Review**

1. Creating the manifest files for deploying an application in the Kubernetes cluster

## **Pod**

A Pod is the smallest deployable unit in Kubernetes. It encapsulates one or more containers, along with storage resources, a unique network IP, and options for how the containers should run. Pods are ephemeral in nature and are usually managed by higher-level controllers like Deployments.

The files requires are

* Creating Pod.yaml file: it creates the pod in the node within our namespace
* It creates a pod in the instance

The script of pod.yaml file is:

apiVersion: v1

kind: Pod

metadata:

  name: umair-pod

  namespace: umair

  labels:

    app: umair-nginx

spec:

  containers:

  - name: umair-nginx

    image: nginx:latest

    ports:

      - containerPort: 80

1. Then creating a deployment.yaml file so that we can deploy/initiate the application

## Deployment

A Deployment is a controller that provides declarative updates for Pods and ReplicaSets. It allows you to define how many replicas of a Pod should run, ensures high availability, and supports rolling updates and rollbacks.

The script for deployment.yaml manifest file is

apiVersion: apps/v1

kind: Deployment

metadata:

  name: uamir-deployment

  namespace: umair

spec:

  replicas: 4

  selector:

    matchLabels:

      app: ipl

  templates:

    metadata:

      labels:

        app: ipl

    spec:

      containers:

        - name: c-1

          image:

          ports:

            - containerPort: 80

1. Then creating a service.yaml file

## Service

A Service is an abstraction that defines a logical set of Pods and a policy to access them. Services provide stable networking and allow communication between different components of an application. Types include ClusterIP, NodePort, and LoadBalancer.

* The main reason to create this file is it exposes the application in different forms as required like Load balancer, Cluster IP etc…

The script of service.yaml file is

apiVersion: v1

kind: Service

metadata:

  name: umair-service

  namespace: umair

spec:

  selector:

    app: ipl

  ports:

    - protocol: TCP

      port: 80 #svc port number and its our wish

      targetPort: 3000 # container port number and we shouldn't change it

  type: LoadBalancer

1. Then creating a replica.yaml file

* The main reason to create this file is if any pod accidentally gets failed then this automatically creates a new pod immediately so that there is no failure of deployment

The script of replica file is

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: umair-replica

  namespace: umair

  labels:

    app: umair-nginx-rs

spec:

  replicas: 3

  selector:

    matchLabels:

      app: umair-nginx-rs

  template:

    metadata:

      labels:

        app: umair-nginx-rs

    spec:

      containers:

        - name: umair-nginx-rs

          image: nginx:latest

          ports:

          - containerPort: 80

1. **ConfigMap**

A ConfigMap is used to store non-confidential configuration data in key-value pairs. It allows applications to be configured without rebuilding container images.

* The script of config.yaml file is

apiVersion: v1

kind: ConfigMap

metadata:

name: umair-config

namespace: umair

data:

APP\_NAME: "Retail-App"

APP\_ENV: "development"

DB\_HOST: "mongodb"

DB\_PORT: "27017"

## Secret

A Secret is similar to a ConfigMap but is designed to store sensitive information like passwords, tokens, or keys. Data is encoded in base64 format.

The script to secret.yaml is

apiVersion: v1

kind: Secret

metadata:

  name: app-secrets

  namespace: umair

type: Opaque

stringData:

  MONGODB\_URI: "REPLACE\_WITH\_YOUR\_MONGODB\_URI"

  SESSION\_SECRET: "umair-secret"

**Summary**

Kubernetes resources such as Pods, Deployments, Services, ConfigMaps, Secrets, PVs, PVCs, Namespaces, and Ingresses provide the building blocks for deploying, scaling, and managing containerized applications. It handles the deployment of application in a smooth and automated way.