**Kubernetes manifest files**

**Pod:** smallest deployable unit.

**Pod.yaml**

apiVersion: v1

kind: Pod

metadata:

name: ankita-pod

namespace: ankita

labels:

app: ankita

spec:

containers:

- name: ankita-container

image: ankita1809/ankita:v1

ports:

- containerPort: 80

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

labels: key-value pair attached to object used for selection purpose.

**Spec** it is the specification. It is like a heart of a yaml.

Containers: list of containers.

name container name inside the pod.

Image: docker image to run.

**Ports** expose the container port.

**Commands of pod**

kubectl apply -f pod.yaml

To run the yaml file.

kubectl get pods

Show all the pods.

kubectl get pods -n ankita

Show all the pods running in the specific namespace.

kubectl get pods -o wide -n ankita

give the node and ip details of pods running in a specific namespace.

kubectl describe pod ankita-pod -n ankita

give the more detailed info about the pods running in the specific namespace.

kubectl logs ankita-pod -n ankita

show the logs of the pod.

kubectl exec -it ankita-pod -n ankita

opens a shell inside the container of pod.

kubectl delete pod Ankita-pod -n ankita

it will delete the pod in a specific namespace.

**Deployment**

It manages the set of pods to run an application.

**deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: ankita-deployment

  namespace: ankita

spec:

  replicas: 2

  selector:

      matchLabels:

        app: ankita

  template:

    metadata:

      labels:

        app: ankita

    spec:

      -  containers:

         name: ankita-container

         image: ankita1809/ankita:v1

           ports:

             -  containerPort: 80

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

Here the resource type is deployment.

**Metadata** stores the information about the resource.

name: name of the deployment.

namespace: logical grouping of resources.

**Spec** it is the specification. It is like a heart of a yaml.

replicas defines number of pod replicas to run.

selector defines which pod belongs to which replicaset.

Template is like blueprint for pods.

Containers list of containers.

name container name inside the pod.

Image: docker image to run.

**Ports** expose the container port.

**Commands in the deployment**

kubectl get deployments

Show all the deployments.

kubectl get deployments -n ankita

Show all the deployments running in the specific namespace.

kubectl get deployments -o wide -n ankita

give the node and ip details of deployments running in a specific namespace.

kubectl describe deployment ankita-deployment -n ankita

give the more detailed info about the deployments running in the specific namespace.

kubectl delete deployment ankita-deployment -n ankita

it will delete the deployment in a specific namespace.

**Replicaset**

It ensures that a specified number of Pod replicas are running at any given time.

**Replicaset.yaml file**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: ankita-rs

  namespace: ankita

  labels:

    app: ankita

spec:

  replicas: 2

  selector:

    matchLabels:

      app: ankita

  template:

    metadata:

      labels:

        app: ankita

    spec:

      containers:

        - name: ankita-container

          image: ankita1809/ankita:v1

          ports:

          -  containerPort: 80

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

Here the resource type is replicaset.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

**Spec** it is the specification. It is like a heart of a yaml.

replicas defines number of pods replicas to run.

selector defines which pod belongs to which replicaset.

Template is like blueprint for pods.

Containers: list of containers.

Name: container name inside the pod.

Image: docker image to run.

**Ports** expose the container port.

kubectl apply -f rs.yaml

To run the yaml file.

kubectl get rs

Show all the replicasets.

kubectl get rs -n ankita

Show all the replicasets running in the specific namespace.

kubectl describe rs ankita-replica -n ankita

give the more detailed info about the pods running in the specific namespace.

**Service:**

It is a method for exposing a new application that is running as one or more pods in your cluster.

**Types of service in k8s**:

1. ClusterIP :

it will expose the application within a cluster.

You can’t access application outside the cluster.

It is for production use.

**Service.yaml for ClusterIP**

apiVersion: apps/v1

kind: Service

metadata:

name: ankita-service

namespace: ankita

spec:

selector:

app: ankita

ports:

- protocol: TCP

port: 80

targetPort: 80

type: ClusterIP

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

**Spec** it is the specification. It is like a heart of a yaml.

selector will find and connect to the pods.

**Ports** expose the container port.

Protocol: the service uses TCP protocol.

Port: Service will be available on port 80.

targetPort: Traffic will be forwarded to container port 80 inside the Pods.

**Type** defines the type of service we are using.

**Commands in service**

kubectl apply -f ankita

To run the yaml file.

kubectl get svc

Show all the services.

kubectl get svc -n ankita

Show all the services in the specific namespace.

kubectl describe svc ankita-service -n ankita

give the more detailed info about the service in the specific namespace.

kubectl delete svc ankita-service -n ankita

it will delete the service in a specific namespace.

1. NodePort:

It will expose the application on each worker nodes within the cluster and outside the cluster.

It is for testing purpose.

**Service.yaml file for NodePort**

apiVersion: apps/v1

kind: Service

metadata:

  name: ankita-service

  namespace: ankita

spec:

  selector:

    app: ankita

  ports:

    -  protocol: TCP

       port: 80

       targetPort: 80

       nodePort: 32981

  type: NodePort

1. LoadBalancer :

It will expose the application outside the cluster.

It will give one external Ip address for accessing the app outside the cluster.

**Service.yaml file for LoadBalancer**

apiVersion: apps/v1

kind: service

metadata:

  name: ankita-service

  namespace: ankita

spec:

  selector:

    app: ankita

  ports:

    -  protocol: TCP

       port: 80

       targetPort: 80

  type: LoadBalancer

**Secret:**

Secret.yaml

apiVersion: v1

kind: Secret

metadata:

name: ankita-secret

namepace: ankita

type: Opaque

data:

username:

password:

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

Type defines the type of secret.

Data defines the information which should be in encoded form.

**CofigMap:**

configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: ankita-configmap

namespace: ankita

data:

DB\_HOST: "mydatabase"

DB\_PORT: "27071"

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

Data defines the information.

**StorageClass:**

sc.yaml

apiVersion:

kind: storageClass

metadata:

name: ankita

namespace: ankita

provisioner: ebs.csi.aws.com

volumeBindingMode:

reclaimPolicy: Delete

parameters:

type: gp2

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

Provisioner specifies which volume plugin should be used to provision the storage.

volumeBindingMode tells k8s to wait until a Pod using this storage is scheduled before binding the volume.

Reclaimpolicy specifies what happens to the storage when the associated PersistentVolumeClaim is deleted.

Parameter specific parameters for the provisioner.

**PVC:**

pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: ankita-pvc

namespace: ankita

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

StorageClassName: ankita-sc

**apiVersion** defines the version of api

**kind** defines that which resource type we are creating.

**Metadata** stores the information about the resource.

name: name of the resource.

namespace: logical grouping of resources.

**Spec** it is the specification. It is like a heart of a yaml.

defines what kind of storage you're requesting.

accessModes specifies how the storage can be accessed.

Resources

Requests specifies how the storage can be accessed.

StorageClassName specifies specifies which StorageClass to use when provisioning the volume.