Cluster contains multiple nodes but there will be only one master node.

**kubectl version**: to get version of kubernetes

**POD**

**kubectl run <pod\_name> --image=<image\_name>** -> it will run a new pod with image.

**kubectl run nginx --image=nginx** -> it will run a new pod called nginx with image as nginx.

**kubectl get pods** -> to get all the pods

**kubectl get pods -o wide** -> to get more details for the pods

**kubectl describe pod <pod-name>** -> it will give us the detailed info about the specific pod

Kubernetes yaml file must contain these fields -> apiVersion, kind, metadata, spec

|  |  |
| --- | --- |
| **kind** | **version** |
| Pod | v1 |
| Service | v1 |
| ReplicationController(deprecated) | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |

nginx-pod.yaml

apiVersion: v1

kind: Pod

metadata:

  name: nginx

  tier: frontend

spec:

  containers:

    - name: nginx

      image: nginx

**kubectl apply/create -f nginx-pod.yaml** -> to create a pod with this yaml file.

**kubectl describe pod nginx** -> to give more description of the nginx pod

**kubectl edit pod nginx** -> it will open a vi editor where we can change the pod definition file also this is a in memory pod definition file which is maintained by Kubernetes.

**kubectl delete pod nginx** -> to delete the nginx pod

**kubectl delete --all pods** -> delete all the pods

**kubectl run redis --image=redis --dry-run=client -o yaml > redis-pod.yaml** -> It will not create any pod rather it’s a imperative style of writing definition file where a pod definition file will created with the necessary fields.

**ReplicaSet**

ReplicaSet is a group of same pods where we can scale in(reduce) or scale out(increase) the number of the pods.

**kubectl create/apply -f <replicaset-definition.yaml>** -> it will create a replicaset from the definition file.

**kubectl get replicateset** -> to get all the replicaset in the default namespace

**kubectl describe replicaset** -> to get all the replicaset in the default namespace

**kubectl delete replicaset <replicaset-name>** -> to delete the replicaset

nginx-replicaset.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: nginx-replicaset

  labels:

    type: frontend

spec:

  selector:

    matchLabels:

      name: nginx-pod

      type: frontend

  replicas: 1

  template:

    metadata:

      labels:

        name: nginx-pod

        type: frontend

    spec:

      containers:

        - name: nginx

          image: nginx

labels under template section and mathLabels under selector should be same. That is how replicate set identifies the pod and controls the number of the pods. If we try to delete any pod or anyhow any pod got crashed then Kubernetes automatically brings another pod in.

Scale commands ->

**kubectl replace -f nginx-replicaset.yaml** -> replace the previous nginx-repicaset with the current replicas given in the definition file

**kubectl scale --replicas=6 -f nginx-replicaset.yaml** -> it will override the replicas given in the yaml

**kubectl edit replicaset nginx-replicaset** -> it will open an editor and show the current configuration of the replicaset then we can easily scale out.

**kubectl scale --replicas=6 replicaset nginx-replicaset** -> it will scale out an existing replicaset

**Deployment**

Kubernetes deployment create creates one deployment kind of object

**kubectl get deployments** -> It will fetch all the deployments

**kubectl describe deployments <deployment-name>** -> It will give us the description of the specific deployment

**kubectl get all** -> To get the resources like pod, replicasets, services, deployments

**kubectl create deployment http-frontend --image=httpd:2.4.alpine** -> it will create one deployment named ad http-frontend and its image will be httpd:2.4.alpine and count of the pod will be 1.

**kubectl scale deployment --replicas=3 httpd-frontend** -> it will scale the deployment and change the number of the pods

**kubectl create -f nginx-deployment.yaml --record** -> It will create a deployment and also record all the changes of the deployment for rollout history.

If we want to revert to previous version, we must add this record flag

**Update and roll back ->**

**kubectl rollout status deployment/nginx-deployment** -> it will give the current rollout status of the deployment. When we change the replicas or the image at that time it will give the information.

**kubectl rollout history deployment/nginx-deployment** -> It will give us all the rollout history of the Kubernetes deployment rollout

For rollout Kubernetes has 2 types of strategy

1. Recreate strategy
2. Rolling strategy

Kubernetes deployment object creates another replicaset and then it will add the pods into the replicaset

**kubectl rollout undo deployment/nginx-deployment** -> It will revert the latest changes back to the previous version.

Changes to the existing deployment:

**kubectl set image deployment nginx-deployment nginx:old-image=nginx:new-image --record** -> It will change the image of the deployment of the nginx deployment and will record it

**kubectl edit deployment nginx-deployment --record** ->it will open vi editor and open the current configuration of the nginx-deployment

**Service**

For accessing the pods from the outside of the container we need service.

There are three types of services.

1. NodePort
2. ClusterIP
3. LoadBalancer

NodePort can range from 30000 to 32767. The work of the NodePort is to listen to a particular port and forward it to another node.

#nginx-service.yaml

apiVersion: v1

kind: Service

metadata:

  name: nginx-service

spec:

  type: NodePort

  ports:

    - port: 80

      targetPort: 80 #where the pod will listen (for nginx its 80)

      nodePort: 30008 #In this port the service will be accesible

  selector:

    app: nginx-pod #to connect with specific pods via pod’s label

**kubectl create -f nginx-deployment.yaml** -> for creating nginx-service with nginx-deployment.yaml

**kubectl get services** -> To get all the services

**kubectl describe service <service\_name>** -> To get details of the specific service

**minikube service nginx-service --url** -> It will print the service url

**ClusterIP**: Internal communication of pods. Service definition is almost same as the NodePort. Here type is **ClusterIP**. **TargetPort** is where the backend is exposed, and **Port** is where the service is exposed.

**LoadBalancer**: With the NodePort service we can make external facing application available on the port of the worker nodes.

Let’s say we have four cluster and one each server there are one frontend app deployed. With NodePort we can make external traffic forwarded to frontend pod but again for that we will have 4 urls for 4 services.

So need to have a loadbalancer here. We can have an external VM where nginx is deployed and then it will loadbalance 4 urls but it will be a complicated thing to manage.

So we can use the inbuilt loadbalancer of different cloud platforms like Azure,GCP or AWS.

Imperative style of creating service:

**kubectl expose deployment nginx-deployment --name=nginx-service --target-port=80 --port=80 --type=NodePort** -> It will create a service named as nginx-service of type NodePort with specific port and targetPort and also it will match labels of the deployment of nginx-deplpyment and NodePort will be be assigned randomly in the range of 30000 to 32767

**kubectl expose deployment nginx-deployment --name=nginx-service --target-port=80 --port=80 --type=NodePort --dry-run=client -o yaml > nginx-service.yaml** -> It will do just the same as previous just that it will save all the configurations in the nginx-service.yaml

If we have a connection like this:

1. Voting-app -> frontend app for gathering the votes which will save the votes in redis
2. Result-app -> frontend app for showing the votes which will fetch votes from postgres
3. Redis -> save the votes from in a in memory store
4. Postgres -> save the votes in relational db
5. Worker-app -> It will constantly fetch the vote count from redis and constantly update the vote count in postgres

So, our setup will be like this:

1. Deploy the pods/deployment
2. Create the ClusterIP service for Redis and postgres
3. Create the NodePort service for Voting-app and Result-app

**NameSpace**

NameSpace is a way to segregate different resources like dev,qa,prod etc. Default namespace is default.

#dev-namespace.yaml

apiVersion: v1

kind: NameSpace

metadata:

  name: dev

**kubectl create -f dev-namepace.yaml** -> to create a namespace

**kubectl get pods --namespace=dev** -> To get all the pod inside dev namespace

If we have to set the dev namespace permanently then we can keep it inside the **KubeConfig**

**Kubectl config set-context $( kubectl config current-context ) --namespace=dev**

To limit the resources using in a specific namespace we can use resource quota

**kubectl get ns** -> to get all the namspaces

**kubectl get ns --no-headers | wc -l** -> To get count of the namespaces

In pod-definition.yaml in metadata we can add namespace to mention the namespace where the pod will be deployed

**kubectl -n dev get pods --no-header** -> to get the pods in dev namespace

# nginx-pod.yaml

apiVersion: v1

kind: Pod

metadata:

  name: nginx-pod

**namespace: dev**

  labels:

    app: nginx-pod

spec:

  containers:

    - name: nginx

      image: nginx

In which namespace the nginx pod is deployed?

Ans: **kubectl get pods --all-namespaces | grep nginx**

In the same namespace we can connect to another pod via pod name but to connect with other resources in another namespace we have to maintain we proper format.

**resource-name.namespace.resource-type.domain**

example: **db-service.dev.svc.cluster.local**

db-service is the name of the resource, dev is the namespace, svc is the resource type, cluster.local is he domain

**Some of the imperative style command**

**--dry-run** -> by default all the command is run with this --dry-run. As soon as the command is run the resource will be created.

**--dry-run=client** -> It will not create the resource rather it will check the whole command and tell us that the command is correct or not.

-**o yaml > resource-definition.yaml** -> this will create the resource definition in an yaml format

**kubectl create deployment nginx --image=nginx --dry-run=client -o yaml > nginx-deployment.yaml** -> It will first check the command is correct or not. If it is correct then will create one nginx-deployment file with the configuration given

**kubectl expose pod redis --port=6379 --name=redis-service --dry-run=client -o yaml > redis-service.yaml** -> it will create a redis-service yaml with the configuration given

**kubectl create service clusterip redis-service --tcp=6379:6379 --dry-run=client -o yaml > redis-service.yaml** -> it will create a dry run of the redis service and save it in redis-service.yaml

**kubectl expose pod nginx-pod --port=80 --name=nginx-service --type=NodePort --dry-run=client -o yaml > nginx-service.yaml** -> It will expose pod named as nginx pod type of NodePort4

**kubectl create service nodeport nginx-service --tcp=80:80 --node-port=30080 --dry-run=client -o yaml > nginx-service.yaml** -> It will create a service of nodeport with port 80 and tagetport 80 with nodeport 30080

kubectl expose command automatically use the pods labels as the selctors but we can not specify the nodeport. We must add that in the definition file then we can add the nodeport.

Kubectl create service command will not use the pod labels as selectors instead it will assume selectors as **app: service-name** and we can not pass selector in the definition file.

**Commands and args**

In docker we have **ENTRYPOINT** and **CMD** for giving command line arguments, but we can override that using **--entrypoint** and the **extra parameters** passed in the docker run command. Same way we can override the existing command in the pod definition file with **command** and **args. ENTRYPOINT** will associated with **command** and **CMD** will be associated with **args.**

Let’s say we have a dockerfile like this

name: ubuntu-sleeper

image: ubuntu-sleeper

command: [“sleeper2.0”]

args: [“100”]

#ubuntu-sleeper dockerfile

FROM UBUNTU

ENTRYPOINT [“sleep”]

CMD [“10”]

The right-side pod definition file is same as this docker run command:

**docker run --entry-point=sleeper2.0 ubuntu-sleeper 100**