Kubernates: this is also container orchestration tool similar to docker swarm

Node: All the individual machine present in the k8 networks are called as nodes

Cluster: It is combinations of nodes

Master node manages the cluster

Slave node execute the work load

Master node has an application called API server

Slave node has an application called kebelet

Kubeadm: This is tool for setting up master and worker on different machines

Kubectl: is used for running kubernates commands

Pod: pod is the single instance of application, k8 does not deploy containers directly on the workers instead it uses a pod within the pod it can run single container or multiple containers ,kubctl command is use for managing the pod

Advantages: Load balancing the containers

Scaling of containers

Performing rolling updates

Handling failover scenarios

To create kubernates clusters:

1. Play-with –k8s.com
2. Katacoda

1.open kacoda-go to k8-click on start course-go to launch multimode cluster kubadm-click on start scenario-login using github account-follow the all steps from step 1 to step 3-istallation is over.

**Commands:**

Use case:

1. Start nginx as pod with 1 replica and later delete the deployment

Kubectl run –image=nginx nginx-app –port=80

1. To see the all pods running

Kubctl get pods

1. To delete the complete deployment

Kubctl delete deployment nginx-app

**Load Balancing:**

Use case

1. start tomcat with 2 replicas in k8 cluster

Kubctl run –image=tomcat tomcat-app –replicas=2 –port==8080

1. To see the status of tomcat in cluster

Kubctl get pod

1. To delete the deployment permanatly

Kubctl delete deployment tomcat-app

Usecase:

1. Start mysql with 2 replicas

Kubctl run –image=mysql mysql-app –env MYSQL\_ROOT\_PASSWORD=intelliq –replicas=2

1. To get the information about 2 replicas of mysql

Kubctl get pods

1. To delete the deployment

Kubctl delete deployment mysql-app

Creating pod using YML files:

These yml files contains 4 top level fields

1. **apiVersion**: This describes the version of k8 api that is use to create this object
2. **kind:** this is represent kind of k8 object that we want to create

kind :version

POD v1

Service v1

ReplicaSet apps/v1

Deployment apps/v1

1. **Metadata**: data about the current k8 object is stored here

Ex: name ,lable etc

This is stored in the form of dictionary object

1. **Spec**: the exact specifications related to the docker image and container are stored here

Create the pod definition file for starting nxing with 1replica:

Vim pod-definition.yml

apiVersion: v1

kind: pod

metadata:

name: nginx-pod

labels:

app: myapp

type: webserver

spec:

containers:

* name: mynginx

image: nginx

1. To deploye the above yml file

Kubctl create –f pod-definition.yml

1. To see the list of pods

Kubctl get pods

1. To delete th pods created using above file

Kubctl delete –f pod-definition.yml

Use case2: starting the mysql pod in kubernates cluster

Vim pod-definition.yml

apiVersion: v1

kind: pod

metadata:

name: mysql-app

labels:

name: mysql

spec:

containers:

* name: mybd

image: mysql:5

env:

name: MYSQL\_ROOT\_PASSWORD

value: intelliq

1. To start the above file

Kubectl create -f pod-definition2.yml

Init 0 used for shutdown the service

Use case:

create pod definition filefor starting the jenkins and port maping

vim pod.definition.yml

apiVersion: v1

kind: pod

meatadata:

name: app-jenkins

labels:

ci: cd

spec:

containers:

- name: myjenkins

image: jenkins

port:

-containerPort: 8080

hostPort: 6060

**Replication controller:**

if you want to perform load balancing directly from the level of pod definition

yml file, we can use replication controller as kind,replication contrller uses key called template in the spect section

where it creates link btw metadata for individual pods with metadata of entire

deployment

usecase:

creare pod definition file for starting tomcat with 3 replicas

vim pod-definition1.yml

apiVerion: v1

kind: ReplicationController

metadata:

name: app-tomcat

spec:

replicas: 3

template:

metadata:

name: mytomcat

labels:

app: tomcat

spec:

containers:

name: tomcat1

image: tomcat

port:

- containerPort: 8080

hostPort: 7070

**ReplicaSet:**

This is an object which is used in kubernates when we want scale

the services or pod on runtime

replicationset is defined with fields which include selector that specify

how to idenfy the pods that can aquire, the number of required replicas

that are required for maintain desired state

it then create or delete the pods until the required count is reached

once replicaset object is created we can scale it up or down based on

our requirement.

Use case:

Staring httpd with 3 replicas later scale the number of replicas to 6

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: frontend

labels:

app: guestbook

tier: frontend

spec:

replicas: 3

selector:

matchLabels:

tier: frontend

templete:

metadata:

labels:

tier: frontend

spec:

containers:

- name: webserver

image: httpd

ports:

- containerPort: 80

hostPort: 7070

To create above rc.definition file

kubectl create -f re-definition.yml

To see if 3 replicas are running with httpd service

kubctl get pods

To scale the number of replicas from 3 to 6

kubctl scale --replicas=6 -f rc-definition.yml

To delete the deployement

kubectl delete -f rc-definition.yml

**Deployment**: is higher level object then replicaset which can be used not

only for load balancing and scaling but it can slaos be prforming rolling update

of the services running into the kubernates cluster

Use case:

create a deployment yml file to run nginx 1.7.9.version with 3 replicas

later perform rolling update to nginx 1.9.1 version