**Kubernetes**

- Orchestration Platform

- To manage containers

- Developed by Google using Go language

- Google donated K8S to CNCF

- K8S first version released in 2015

- It is free & Open source

Note: Kubernetes will use Docker internally. Using K8S we will manage our Docker containers.

**Docker Swarm Vs K8S**

- Docker Swarm doesn't have Auto Scaling (Scaling is manual process)

- K8S supports Auto Scaling

- For Production deployments K8S is highly recommended

- Kubernetes is replacement for Docker Swarm

Auto Scaling : Increasing and Decreasnig containers count based on incoming requests for our app.

**What is Cluster**

- Group Of Servers

- Master Node(s)

- Worker Node(s)

- DevOps Enginner / Developer will give the task to K8S Master Node

- Master Node will manage worker nodes

- Master Node will schedule tasks to worker nodes

- Our containers will be created in Worker Nodes

- Using Cluster we can achieve High Availability

**Kubernetes Architecture**

- Control Plane / Master Node / Manager Node

- Api Server

- Schedular

- Control Manager

- ETCD

- Worker Node (s)

- Pods

- Containers

- Kubelet

- Kube Proxy

- Docker Runtime

**How to communicate with K8S control plane ?**

- Kubectl (CLI tool)

- Web UI Dashboard

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

**Kubernetes Architecture Components**

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

Control Plane : Control Plane is called as Master Node (Responsible to handle k8s related work)

Worker Nodes: Responsible to execute our application as PODS.

=> API Server : It is responsible to handle incoming requests of Control Plane ( to deploy our applications)

=> Etcd : It is an internal database in K8S cluster, API Server will store requests / tasks info in ETCD

=> Schedular : It is responsible to schedule pending tasks available in ETCD. It will decide in which worker node our task should execute. Schedular will decide that by communicating with Kubelet.

=> Kubelet : It is a worker node agent. It will maintain all the information related to Worker Node.

=> Conroller-Manager : After scheduling completed, Controller-Manager will manage our task execution in worker node

=> Kube-Proxy : It will provide network for K8S cluster communication (Master Node <---> Worker Nodes)

=> Docker Engine : To run our containers Docker Engine is required. Containers will be created in Worker Nodes.

=> Container : It is run time instance of our application

=> POD : It is a smallest building block that we will create in k8s to run our containers.

Note: Docker Containers will run inside POD.

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

**Kubernetes Cluster Setup**

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

1) Self Managed Cluster ( We will create our own cluster )

a) Mini Kube ( Single Node Cluster )

b) Kubeadm (Multi Node Cluster)

2) Provider Managed Cluster (Cloud Provide will give read made cluster) ---> Charges applies

a) AWS EKS

b) Azure AKS

c) GCP GKE

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

**Minikube Cluster setup on windows OS**

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

1) Download and install Docker Desktop software in Windows

( URL : https://docs.docker.com/desktop/install/windows-install/ )

2) Download and install Minikube ( URL : https://minikube.sigs.k8s.io/docs/start/)

3) Download and install Kubectl (URL :

https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/ )

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

**Kubernetes Components**

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

1) POD

2) Services

3) Namespaces

4) Replication Controller

5) Replication Set

6) DaemonSet

7) Deployment

8) StatefulSet

9) ConfigMap & Secrets

10) IngressController

11) K8S Volumes

12) HELM Charts

13) Grafana & Promethues

14) ELK Setup (Log Aggregation)

15) RBAC

16) K8S Web Dashboard

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

**What is POD ?**

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

=> POD is a smallest building block in k8s cluster

=> In K8S, every container will be created inside POD only

=> POD always runs inside Node

=> POD represents running process

=> POD means group of containers running on a Node

=> We can create multiple PODS on single node

=> Every POD will have unique IP address

=> We can create PODS in 2 ways

1) Interactive Approach

$ kubectl run --name <pod-name> image=<image-name> --generate=pod/v1

2) Declarative Approach ( K8S Manifest YML )

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

**K8S Manifest YML**

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

apiVersion:

kind:

metadata:

spec:

# Display all pods which are created

$ kubectl get pods

# Create PODS using pod-manifest

$ kubectl apply -f <pod-yml>

# Descrie pod ( get more info about pod)

$ kubectl describe pod <pod-name>

# Get pod logs

$ kubectl logs <pod-name>

# Get POD running on which worker node

$ kubectl get pods -o wide

############### Note: PODS we can't access outside. #############

=> We need to expose PODS for outside access using Kubernetes Service concept.

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

**K8S Service**

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

=> Kubernetes service is used to expose PODS outside cluster

=> We have 3 types of K8S Services

1) Cluster IP

2) Node Port

3) Load Balancer

=> We need to write service manifest yml to expose PODS

# Display existing services

$ kubectl get svc

# Create k8s service

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

# Display existing services

$ kubectl get svc

Note: We can see service information and Node Port Number assigned by K8S.

Note: Enable Node Port number in security group of Worker node in which our POD is running.

#Access application in browser

URL : http://node-public-ip:node-port-num/java-web-app/

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

**Working Procedure**

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

1) Write POD Manifest YML

2) Create POD using kubectl apply command

3) Check POD details and POD logs & In which worker node it is running

4) Write Service Manifest YML

5) Create Service to expose PODS

6) Check Service Details and Node PORT number assigned by K8S

7) Enable Node PORT number in Worker Node Security Group

8) Access application using Worker Node IP

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

**Cluster IP**

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

-> When we create PODS, every pod will get unique IP address

-> We can access POD inside cluster using its IP address

Note: PODS are short lived objects, When POD is recreated its ip will be changed so we can't depend on POD IP to access.

-> To expose POD access with in the cluster we can use ClusterIP service.

-> ClusterIP will generate one IP address to access our PODS with in cluster.

Note: ClusterIP will not change when PODS are re-created.

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

**Node Port**

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

=> NodePort service is used to expose pods outside cluster also

=> When we use NodePort service we can specify Node Port number in service manifest file

=> If we don't specify Node Port number then k8s will assign random node port number for our

service.

Node Port Range : 30000 - 32767

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

**Load Balancer**

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

-> It is one type of K8s Service

-> It is used to expose our PODS outside cluster using Load Balancer

-> When we use Load Balancer as service type then one Load Balancer will be created in AWS Cloud.

-> Using Load Balancer URL we can access our application

-> Load Balancer will distribute the traffic to multiple worker nodes in Round Robbin fashion.

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

**POD Life Cycle**

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

-> POD is a smallest building block that we can run in k8s cluster

-> We are using kubectl to send request to control plane to create POD

-> API Server will recieve our request and will store request details in ETCD.

-> Schedular will find un-scheduled PODS and it will schedule in worker nodes

-> Node Agent (kubelet) will see POD schedule and it will fire Docker Engine

-> Docker will engine will run our container

Note: PODS are ephemeral (Short Lived Objects)

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

**K8S Namespaces**

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

-> Namespaces are equal to packages in the Java.

-> Namespaces are used to group our k8s components logically.

app pods ----> ashokit-app-ns

db pods ----> ashokit-db-ns

-> We can create multiple namespaces in k8s cluster

ex: ashokit-app-ns, ashokit-db-ns etc.....

-> Namespaces are logically isolated with each other

Note: When we delete namespace all the components created under that namespace will be deleted.

-> We can get k8s namespaces using below command

$ kubectl get ns (or) $ kubectl get namespace

-> In K8s we will have below namespaces by default

1) default

2) kube-public

3) kube-system

4) kube-node-lease

Note: When we create k8s component without using namespace then k8s will consider 'default' namespace for that.

Note: kube-public, kube-system and kube-node-lease namespaces will be used by k8s cluster.

We should not create our k8s components in these 3 namespaces.

# Command to get k8s components

$ kubectl get all

# Get k8s components of given namespace

$ kubectl get all -n <namespace>

Ex : kubectl get all -n kube-system

It is highly recommended to create our k8s components under custom namespace

# create namespace

$ kubectl create ns ashokitns

=> Always we need to create the PODS using below K8S components/resources

1) ReplicationController

2) ReplicaSet

3) Deployment

4) StatefulSet

5) DaemonSet

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

**ReplicationController (RC)**

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

=> It is a k8s component which is used to create PODS

=> It will make sure always given no.of pods are running for our application.

Note: It will take care of POD lifecycle

Note: When POD is crashed / damaged / deleted then RC will create new pod.

=> We can scale up and scale down PODS count using Replication Controller

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

**ReplicaSet**

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

-> ReplicaSet is replacement for ReplicationController (It is nextgen component in k8s)

-> ReplicaSet is also used to manage POD Lifecycle

-> ReplicaSet also will maintain given no.of pods always

-> We can scale up and we can scale down our PODS using ReplicaSet also

### The only difference between ReplicationController and ReplicaSet is 'selector' ###

=> ReplicationController supports Equality Based Selector

ex:

selector:

app: javawebapp

=> ReplicaSet supports Set Based Selector

ex:

selector:

matchLabels:

app: javawebapp

version: v1

type: backend

===========

**Deployment**

===========

=> Deployment is one of the K8S resource / component

=> Deployment is the most recommended approach to deploy our applications in k8s cluster.

=> Using Deployment we can scale up and we can scaledown our pods

=> Deployment supports Roll Out and Roll Back.

Note: To deploy latest code we have to delete RC & RS then PODS will be deleted and application will be down.

=> We can deploy latest code with Zero Downtime using "Deployment"

1) Zero Downtime Deployment

2) Rollout and Rollback

3) AutoScaling

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

**Deployment Strategies**

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

=> We have below Deployment stratagies

1) Re Create

2) Rolling Update

=> ReCreate means it will delete all the existing pods and it will create new pods (downtime will be there)

=> RollingUpdate strategy means it will delete the pod creates new pod one by one.

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

Blue - Green Deployment

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

-> It is one of the approach to deploy application

1) less risk

2) zero downtime

3) easy rollbacks

4) semless user experience

=> We need to create below manifest ymls for blue-green deployment

1) blue-deployment.yml

2) live-service.yml

3) green-deployment.yml

4) pre-prod-service.yml

Step-1) Create blue-deployment

Step-2) Expose Blue PODS using Live-Service

Step-3) Access application using Live Service (Blue PODS will run)

Step-4) Clone git repo (https://github.com/ashokitschool/java\_web\_docker\_app)

Step-5) Modify Code + Build Project using Maven + Create Docker Image + Push Image to Docker Hub

Step-6) Create Green-Deployment with latest image

Step-7) Expose Green PODS using Pre-Prod-Service

Step-8) Access application using pre-prod Service (green PODS will run)

Note: Notedown Live Service URL and Pre-Prod-Service URL

Step-9) Access both URLS and see the difference

Live Service URL : http://3.109.202.11:30785/java-web-app/

Pre-Prod-Service URL : http://3.109.202.11:31785/java-web-app/

Step-10) Modify the selector in live-service from v1 to v2 and apply that using kubectl

Step-11) Access live service url (latest code should be accessible)

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

**Config Map & Secrets**

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

=> For every application multiple environments will be available for testing purpose

a) DEV

b) SIT

c) UAT

d) PILOT

=> Once appilcation testing is completed in all above environments then it will be deployed into Production environment (Live Environment).

Note: For every envionment, application properties will be different

a) Database Properties

b) Kafka properties

c) SMTP properties

d) Redis properties etc...

### We shouldn't hardcode application properties ###

=> Config Map & Secret concepts are used to avoid hard coded properties in the application

=> Config Map is used to store data in key-value (non-confidential)

=> Config Map allows us to de-couple application properties from Docker images so that our application can be deployed into any environment without making any changes for our Docker image.

=> Secret is also one of the k8s resource

=> Secret is used to store confidential data in key - value format (ex: pwd)

=> Secret is used to store confidential data in encoded format.

URL To encode : https://www.base64encode.org/

### Note: ConfigMap & Secrets will make docker images as portable. ###

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

**Stateless application vs Stateful Application**

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

=> Stateless applications can't store the data permanentley. For every request they will get new

data and it will process that data.

Every request will be treated as new request.

Ex: Node JS app, Nginx app, Boot app etc....

=> Stateful applications means they will store the data permanentley and they will keep track of

data.

Ex: MySQL, Oracle, Postgres etc....

=> To run stateful containers in k8s cluster we will use StatefulSet concept.

=> StatefulSet will assign a sticky identity for each pod starting from zero ( 0 )

=> In Stateful set primary and secondary pods will be created.

=> Once primary pod is created then by copying primary pod data secondary pod will be created

Note: New POD will be created by copying previous pod data. If previous pod is pending state then new pod will not be created.

=> In StatefulSet, pods will be deleted in reverse order (last to first)

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

**What is the difference between Deployment and StatefulSet ?**

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

-> Deployment will create the PODS with random ids

-> Deployment will scale down pods in random order

-> Deployment pods are stateless pods

-> StatefulSet will create the PODS with sticky identity

-> StatefulSet will scale down pods in reverse order

-> StatefulSet pods are statefull

Note:

-> For application deployment we will use 'DEPLOYMENT'

-> For database deployment we will use 'STATEFULSET'

#### Stateful Set Manifest YML Updated in Repo :

https://github.com/ashokitschool/kubernetes\_statefulset\_ymls.git ####

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

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

**DaemonSet**

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

-> It is k8s resource to create PODS

-> DaemonSet will create one pod in each worker node

-> If new node is created new DaemonSet is created auto

-> When we add node to cluster then POD will be created, if we remove node from cluster then

POD will be deleted..

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

When to use DaemonSet

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

1) Logs Collector

2) Node Monitoing

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

**HELM Charts**

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

-> HELM is a package manager for K8S Cluster

-> HELM is used to install required softwares in k8s cluster

-> HELM will maintain Charts in chart repository

-> Chart means collection of configuration files organized in a directory

##################

**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

-> check do we have metrics server on the cluster

$ kubectl top pods

$ kubectl top nodes

# check helm repos

$ helm repo ls

# Before you can install the chart you will need to add the metrics-server repo to helm

$ helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/

# Install the chart

$ helm upgrade --install metrics-server metrics-server/metrics-server

$ kubectl top pods

$ kubectl top nodes

$ helm list

$ helm delete <release-name>

#######################

**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 container orchestration platform.

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

**Grafana**

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

-> Grafana is a analysis and monitoring tool

-> Grafana is a multi-platform open source analytics and interactive visualization web application.

-> It provides charts, graphs, and alerts for the web when connected to supported data sources.

-> Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore and share dashboards.

Note: Graphana will connect with Prometheus for data source.

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

**Autoscaling**

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

-> It is the process of increasing / decreasing infrastructure or resources based on demand

-> Autoscaling can be done in 2 ways

1) Horizontal Scaling

2) Veriticle Scaling

-> Horizontal Scaling means increasing number of instances/servers

-> Veriticle Scaling means increasing capacity of single system

HPA : Horizontal POD Autoscaling

VPA : Vertical POD Auoscaling (we don't use this)

HPA : Horizontal POD Autoscaler which will scale up/down number of pod replicas of deployment, ReplicaSet or Replication Controller dynamically based on the observed Metrics (CPU or Memory Utilization).

-> HPA will interact with Metric Server to identify CPU/Memory utilization of POD.

# to get node metrics

$ kubectl top nodes

# to get pod metrics

$ kubectl top pods

Note: By default metrics service is not available

-> Metrics server is an application that collect metrics from objects such as pods, nodes according to the state of CPU, RAM and keeps them in time.

-> Metric-Server can be installed in the system as an addon. You can take and install it directley from the repo

########################

**Kubernetes Ingress**

########################

Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster.

-> Deploy two application Into K8S using Service using Cluster IP

$ kubectl apply -f javawebapp.yml

$ kubectl apply -f mavenwebapp.yml

-> Now we have 2 services running in K8S cluster with Cluster IP service. We can't access them outside the cluster.

-> We will use Ingress to provide routing for these two services from external traffic

-> K8S ingress is a resource to add rules for routing traffic from external sources to the services in the k8s cluster

-> K8S ingress is a native k8s resource where you can have rules to route traffic from an external source to service endpoints residing inside the cluster.

-> It requires an ingress controller for routing the rules specified in the ingress object

-> Ingress controller is typically a proxy service deployed in the cluster. It is nothing but a Kubernetes deployment exposed to a service.