# KUBERNETES

**Container Orchestration**- automatically deploying and managing containers

Kubernetes is a container orchestration technology/ is a container orchestration technology used to orchestrate the deployment and management of hundreds and thousands of containers in a clustered environment.

**KUBERNETES ARCHITECTURE**

**Node** – is a machine, physical or virtual on which Kubernetes is installed. A node is a worker machine(also called minion) where containers are launched by Kubernetes.

**Cluster** – a set of nodes grouped together

**Master** – another node with Kubernetes installed and configured as master. The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes.

For ex- when a node fails, master node can move the workload of the failed node to another worker node.

**COMPONENTS**

When installing Kubernetes, the following components are installed

**Kube-API Server** – acts as front end for Kubernetes. The command line interfaces talk to the API Server to interact with the Kubernetes cluster

**ETCD keystore** – key value store used by Kubernetes to store all data used to manage cluster. For ex- when you have multiple nodes and multiple masters in your cluster, etcd stores all that information on all the nodes in the cluster in a distributed manner. Etcd is responsible for implementing locks within the cluster to ensure that there are no conflicts between the masters.

**Kubelet** – is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.

**Container Runtime** – underlying software that is used to run containers- which is docker here

**Controller** - They are responsible for noticing and responding when nodes, containers or end points goes down. The controller makes decisions to bring up new containers in such cases.

Or

the controller is responsible for making decisions to bring up new containers in case any node or container fails/ goes down.

**Scheduler** – responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to nodes.

MASTER vs WORKER NODES

A blue and white screen

Description automatically generated with medium confidence

The worker node or minion as it is also known is where the containers are hosted which is docker containers. And to run docker containers on a system, we need container runtime installed and that’s where the container runtime falls. In this case the runtime is docker.

The master server has the kube API server and that is what makes it a master

The worker nodes have the kubelet agent that is responsible for interacting with master to provide health information of the worker node and carry out actions requested by master on the worker nodes.

All the info gathered are stored in a key value store on the master which is the etcd framework

Kubectl – kube command line tool/kubectl/kube control – is a tool used to deploy and manage applications on a Kubernetes cluster

To get cluster info, to get the status of other nodes in the cluster etc

**kubectl run hello-minikube** – deploy an application on the cluster

**kubectl cluster-info** – view info about the cluster

**kubectl get nodes** – list all nodes as part of the cluster

**PODS**

Pod- A POD is a single instance of an application.

Types of Pod – Single container pod, Multi container pod

Single container pod have single container running inside the pod. Eg - kubectl run <name of pod> --image=<name of the image from registry>

Multi container pods are created using yaml file with the definition of the containers.

apiVersion: v1

kind: Pod

metadata:

name: Tomcat

spec:

containers:

- name: Tomcat

image: tomcat: 8.0

ports:

containerPort: 7500

imagePullPolicy: Always

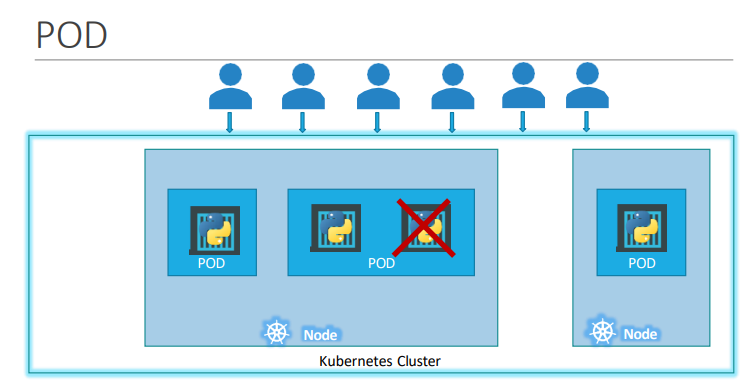
-name: Database

Image: mongoDB

Ports:

containerPort: 7501

imagePullPolicy: Always



Deploy pods

Kubectl run nginx –image=nginx – image nginx is downloaded from docker hub repository and it automatically creates a pod and runs the container in pod

Kubectl get pods - list the pods in a cluster

Kubectl describe pod nginx – gives detailed info on the pod

kubectl get pods -o wide – provides additional info such as IP and on which node it is running

kubectl create deployment nginx --image=nginx – to create a deployment

**YAML (yet another markup language)**

Key value pair

Fruit: Apple

Vegetable: Carrot

Liquid: Water

Array/Lists

Fruits:

* Orange
* Apple
* Banana

Dictionary/Map

Banana:

Calories: 105

Fat: 0.4g

Dictionary is an unordered collection whereas lists are ordered collection.

#- comment

YAML IN KUBERNETES

apiVersion: v1

kind: Pod

metadata: (dictionary)

name: myapp-pod

labels: (dictionary that can have any key value pair)

app: myapp

type: frontend

spec:

containers: (-object -list/array as pods can have multiple containers under it)

* name: nginx-container

image: nginx

kubectl create -f filename.yml / kubectl apply -f filename.yml – to deploy the resources

**apiVersion** – version of the Kubernetes API used to create objects

**kind** – type of object we are trying to create

**metadata**-The metadata is data about the object like its name, labels and annotations etc. Data that helps uniquely identify the object, including a name string, UID, and optional namespace

**spec**-specification, What state you desire for the object, spec is a dictionary. It outlines the configuration details and behavior of the resource.

Whats inside the object we are creating.

**Kind-Version**

POD-v1

Service-v1

ReplicaSet-apps/v1

Deployment-apps/v1

Example

apiVersion: v1

kind: Pod

metadata:

name: postgres

labels:

tier: db-tier

spec:

containers:

- name: postgres

image: postgres

env:

- name: POSTGRES\_PASSWORD

value: mysecretpassword

Commands:

kubectl version

kubectl get nodes

kubectl get nodes -o wide

kubectl run nginx --image=nginx

kubectl describe pod newpods-gdm4t | grep -i image

kubectl get pods -o wide

running containers in pod/ total containers in pod – 1/1 -> Ready

kubectl apply -f pod-definition.yaml

apiVersion: v1

kind: Pod

metadata:

name: redis

spec:

containers:

- name: redis

image: redis

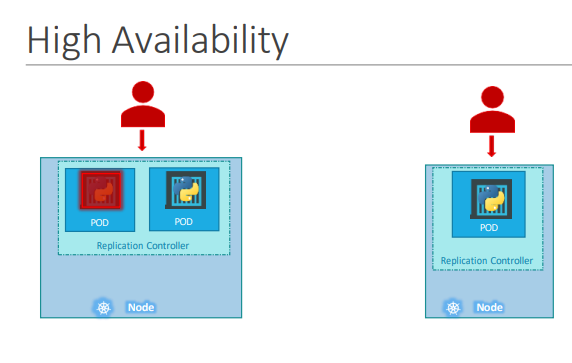
**REPLICATION CONTROLLER**

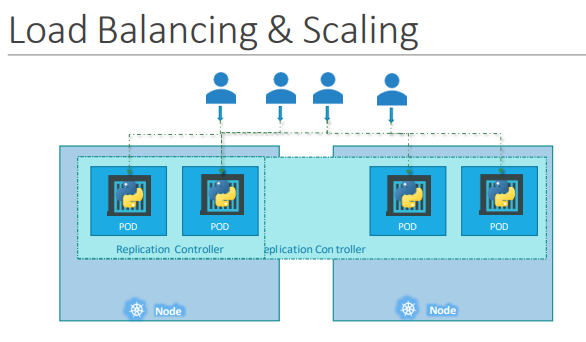
Replication controller helps run multiple instances of single pod in Kubernetes cluster thus providing high availability.

It ensures that the specified number of pods are running at all times even if it is 1 or 100.

Another reason we need replication controller is to create multiple pods to share the load across them.

It helps balance the load across multiple pods on different nodes as well as scale the application when demand increases.





apiVersion: v1

kind: ReplicationController

metadata:

name: myapp-rc

labels:

app: myapp

type: front-end

spec:

template:

metadata: (dictionary)

name: myapp-pod

labels: (dictionary that can have any key value pair)

app: myapp

type: frontend

spec:

containers: (-object -list/array as pods can have multiple containers under it)

* name: nginx-container

image: nginx

replicas: 3

kubectl apply -f pod-defintion.yml

kubectl get replicationcontroller

A screenshot of a computer

Description automatically generated

**REPLICA SET**

New way of replication controller

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: myapp

labels:

app: myapp

type: front-end

spec:

template:

metadata: (dictionary)

name: myapp-pod

labels: (dictionary that can have any key value pair)

app: myapp

type: frontend

spec:

containers: (-object -list/array as pods can have multiple containers under it)

* name: nginx-container

image: nginx

replicas: 3

selector:

matchLabels:

type: front-end

selector section helps replica set identify what pods fall under it.

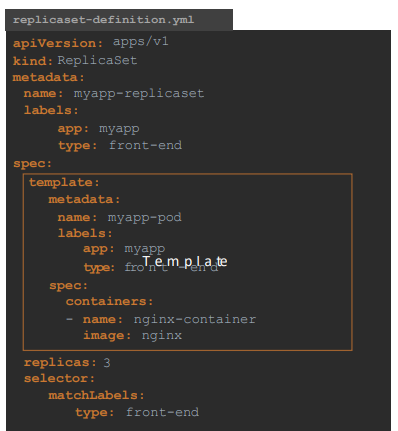
Kubectl apply -f pod-definition.yml

Kubectl get replicaset

You can use replicaset to monitor existing pods if you have hem already created

The role of replica set is to monitor the pods and if any of them fail it will deploy new ones

Labels are used as filter to identify pods among multiple pods



Kubectl replace -f replicaset-definition.yml (to update the replicaset)

or

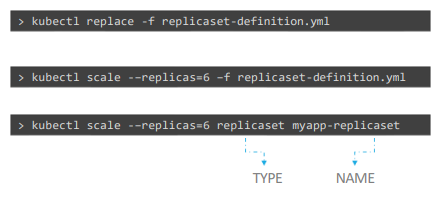
Kubectl scale --replicas=6 replicaset-definition.yml

Kubectl scale --replicas=6 replicaset myapp-replicaset

kubectl delete replicaset myapp-replicaset

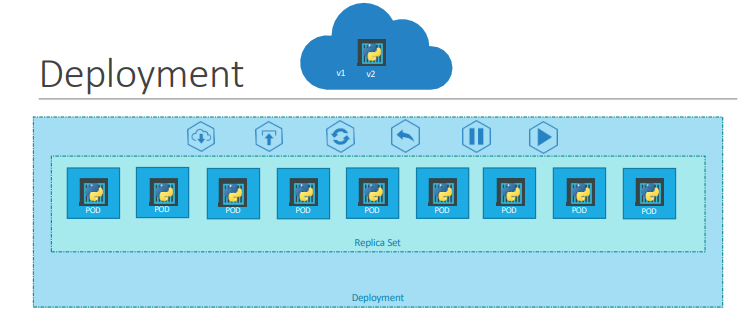
kubectl describe replicaset myapp-replicaset

kubectl edit replicaset myapp-replicaset



**DEPLOYMENTS**

Update the application one by one without affecting the web server – rolling updates



Deployment provides us with the capability to upgrade the underlying instances using rolling updates, undo changes and pause and resume changes as required.

Yaml file is same as ReplicaSet except for kind which is Deployment here. The deployment automatically runs replicaset. The replicaset run pods in turn.

Kubectl create -f deployment-definition.yml

kubectl get deployments

kubectl get replicaset / kubectl get rs

kubectl get pods

kubectl describe deployment deployment-definition.yml

kubectl create deployment httpd-frontend --image=httpd:2.4-alpine --replicas=3

UPDATES AND ROLLBACK IN DEPLOYMENT

Rollout and Versioning

When you first create a deployment, it triggers a rollout, a new rollout creates a new deployment

When the application is upgraded, when the container version is updated to a new one, a new rollout is triggered and a new deployment revision is created named Revision two. This helps us keep track of the changes made to the deployment and enables us to rollback to previous version of deployment.

kubectl rollout status deployment/myapp-dep – to see the status of the rollout

kubectl rollout history deployment/myapp-dep – to show the revisions and history of deployment

Two types of deployment strategies:

1. Recreate-One way to upgrade these to a newer version is to destroy all of these and then create newer versions of application instances. Which means first destroy the five running instances and then deploy five new instances of new applications version.
2. RollingUpdate-Take down the older version and bring up newer version one by one. This way the application doesn’t go down.

To update the image of the application

kubectl set image deployment/myapp-deployment \nginx-container=nginx:1.9.1

A screenshot of a computer program

Description automatically generated

Rollback

Kubernetes deployments allow to rollback to a previous revision

to undo a change – kubectl rollout undo deployment/myapp-deployment

Pod-

apiVersion: v1

kind: Pod

metadata:

name: pod-example

labels:

app: ashitha

spec:

containers:

- name: nginx

image: nginx

- name: postgres

image: postgres

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

Replica Set-

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: replica-example

labels:

app: ashitha

spec:

template:

metadata:

name: pod-example

labels:

app: ashitha

spec:

containers:

- name: nginx

image: nginx

- name: postgres

image: postgres

replicas: 3

selector:

matchLabels:

app: ashitha

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

Deployment-

apiVersion: apps/v1

kind: Deployment

metadata:

name: replica-example

labels:

app: ashitha

spec:

template:

metadata:

name: pod-example

labels:

app: ashitha

spec:

containers:

- name: nginx

image: nginx

- name: postgres

image: postgres

replicas: 3

selector:

matchLabels:

app: ashitha

**NETWORKING IN KUBERNETES**

IP Address is assigned to Pod, each pod has its own internal IP address

by using simple routing techniques the cluster networking enables communication between the different PODs or Nodes to meet the networking requirements of kubernetes. Thus all PODs can now communicate to each other using the assigned IP addresses

**KUBERNETES SERVICES**

Kubernetes services ia an object that enables communication between various components within and outside of the application. It helps us connect applications together with other applications or users. It is services that enable connectivity between these groups of pods.

NodePort – where the service makes an internal port accessible on a port on the node.(helps in mapping a port on the node to a port on the pod)

ClusterIP – the service creates a virtual IP inside the cluster to enable communication between different set of pods.

LoadBalancer – where it provisions a load balancer for the application

A blue and white rectangle with a circle in it

Description automatically generated

NODEPORT

A screenshot of a computer

Description automatically generated

TargetPort – port on the pod

Port – port on the service

NodePort – Port on the node

apiVersion: v1

kind: Service

metadata:

name: webapp

spec:

type: NodePort

ports:

- targetPort: 80

port: 80

nodePort: 30008

selector: (take it from the pod definition-spec file to link the service to the pod)

app: myapp

type: frontend

kubectl create/apply -f service-definition.yml

kubectl get services

CLUSTERIP – default service created by Kubernetes on launch

Each service gets an IP and name assigned to it inside the cluster and that is the name that should be used by other PODs to access the service. This type of service is known as ClusterIP

apiVersion: v1

kind: Service

metadata:

name: backend

spec:

type: ClusterIP

ports:

- port: 80

targetPort: 80

selector:

app: myapp

type: backend

kubectl create/apply -f service-definition.yml

kubectl get services/svc

LOAD BALANCER

apiVersion: v1

kind: Service

metadata:

name: webapp

spec:

type: LoadBalancer

ports:

- targetPort: 80

port: 80

nodePort: 30008

selector:

app: myapp

type: frontend

Kubernetes installation on Rhel 8

Install Kubernetes from the commands given in official page and install minikube.

<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

<https://minikube.sigs.k8s.io/docs/start/?arch=%2Flinux%2Fx86-64%2Fstable%2Frpm+package>