Kubernetes :

Kubernetes is a container management technology developed in Google lab to manage containerized applications in different kind of environments such as physical, virtual, and cloud infrastructure. It is an open source system which helps in creating and managing containerization of application.

Kubernetes in an open source container management tool hosted by Cloud Native Computing Foundation (CNCF).

Kubernetes comes with a capability of automating deployment, scaling of application, and operations of application containers across clusters. It is capable of creating container centric infrastructure.

Features of Kubernetes:

- Continues development, integration and deployment
- Containerized infrastructure
- Application-centric management
- Auto-scalable infrastructure

One of the key components of Kubernetes is, it can run application on clusters of physical and virtual machine infrastructure. It also has the capability to run applications on cloud.
It helps in moving from host-centric infrastructure to container-centric infrastructure.

Kubernetes - Cluster Architecture

Kubernetes follows client-server architecture. Wherein, we have master installed on one machine and the node on separate Linux machines.

Kubernetes - Master Machine Components:

-API server

-etcd

-controller managment

-scheduler

etcd:(data base in the key value formate)

It stores the configuration information which can be used by each of the nodes in the cluster.

It is a high availability key value store that can be distributed among multiple nodes.

It is accessible only by Kubernetes API server as it may have some sensitive information.

It is a distributed key value Store which is accessible to all.

API Server:

Kubernetes is an API server which provides all the operation on cluster using the API.

API server implements an interface, which means different tools and libraries can readily communicate with it.

Kubeconfig is a package along with the server side tools that can be used for communication. It exposes Kubernetes API.

Controller Manager:

This component is responsible for most of the collectors that regulates the state of cluster and performs a task.

In general, it can be considered as a daemon which runs in nonterminating loop and is responsible for collecting

and sending information to API server. It works toward getting the shared state of cluster and then make changes

to bring the current status of the server to the desired state. The key controllers are replication controller,

endpoint controller, namespace controller, and service account controller. The controller manager runs different

kind of controllers to handle nodes, endpoints, etc.

Different types of controllers:

Replication controller

Endpoint controller

Namespace controller

Service account controller

Scheduler:

This is one of the key components of Kubernetes master. It is a service in master

responsible for distributing the workload. It is responsible for tracking utilization

of working load on cluster nodes and then placing the workload on which resources are

available and accept the workload. In other words, this is the mechanism

responsible for allocating pods to available nodes. The scheduler is responsible for workload utilization and allocating pod to new node.

Kubernetes - Worker Node Components:

Docker:

The first requirement of each node is Docker which helps in running the encapsulated

application containers in a relatively isolated but lightweight operating environment.

Kubelet Service:

This is a small service in each node responsible for relaying information to and from control plane service.

It interacts with etcd store to read configuration details and wright values. This communicates with the master component to receive commands and work.

The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

Kubernetes Proxy Service:

This is a proxy service which runs on each node and helps in making services available to the external host.

It helps in forwarding the request to correct containers and is capable of performing primitive load balancing.

It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well.

It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.

Manifest file :

In Kubernetes, a manifest file is a YAML or JSON file that describes the desired

state of Kubernetes objects, such as pods, deployments, services, and others.

The manifest file is used to create, update, or delete Kubernetes objects.

Syntax:

Manifest files are written in YAML or JSON syntax. YAML is the

preferred format because it's more human-readable and easier to write and edit.

API Version: The manifest file starts with the API version of the Kubernetes object

that it describes. This specifies the version of the Kubernetes API that the object is defined in.

Kind: The "kind" field specifies the type of Kubernetes object that the manifest file describes,

such as Pod, Deployment, Service, ConfigMap, Secret, and others.

Metadata: The metadata section includes information about the Kubernetes object,

such as its name, labels, and annotations.

objects in Kubernetes:

Pods

It is the smallest and simplest basic unit of the Kubernetes application.

This object indicates the processes which are running in the cluster.

Node

A node is nothing but a single host, which is used to run the virtual or physical machines. A node in the Kubernetes cluster is also known as a minion.

Service

A service in a Kubernetes is a logical set of pods, which works together. With the help of services, users can easily manage load balancing configurations.

ReplicaSet

A ReplicaSet in the Kubernetes is used to identify the particular number of pod replicas are running at a given time. It replaces the replication controller because it is more powerful and allows a user to use the "set-based" label selector.

what is pod ..?

It is the smallest and simplest basic unit of the Kubernetes application. This object indicates the processes which are running in the cluster.

A pod is a collection of containers and its storage inside a node of a Kubernetes cluster. It is possible to create a pod with multiple containers inside it. For example, keeping a database container and data container in the same pod.

basic commands used with pods :

kubectl apply -f <file name> : it will create the pod

kubectl get pods <pod name> : it will list all the pods

kubectl describe pod <pod-name> : it will give the details of the pod

kubectl logs <pod-name> <container-name>: it will give the logs of the specific container in the pod

kubectl exec <pod-name> -c <container-name> -- <command> : to run command inside the specific pod

kubectl delete pod <pod-name> : it will delete the specific pod

we can delete multiple pods at a time

kubectl delete pods -l app=backend : this would delete all the pod having backend

Types of pods:

single container pod

multi container pod

we can create a pod in one line that is:

kubectl run tomcat --image = tomcat:8.0

it will create the tomcat pod ,it will take the image from the docker hub and run

we have muilti container pod where we need to write YAML flie for the creating the pod

eg:

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
```

In Kubernetes, a cluster is a set of nodes that run containerized applications. The Kubernetes cluster consists of a Master node and multiple Worker nodes. The Master node manages the overall state of the cluster and coordinates the activities of the Worker nodes.

The Master node is responsible for managing the state of the Kubernetes API server, the etcd database, the scheduler, and the controller manager. The Worker nodes, on the other hand, are responsible for running containerized applications and reporting their status back to the Master node.

To create a Kubernetes cluster, you will need to have at least one Master node and one or more Worker nodes. You can create a cluster using a variety of methods, such as using a cloud provider's managed Kubernetes service, using a Kubernetes installer tool like kubeadm or kops, or using a container orchestration platform like OpenShift.

Once the cluster is up and running, you can deploy your containerized applications to it using Kubernetes manifests, which define the desired state of your applications and their associated resources, such as Pods, Deployments, Services, and ConfigMaps. The Kubernetes scheduler will then schedule the containers to run on the Worker nodes based on their resource requirements and availability.

Kubernetes also provides a wide range of features to manage and monitor the cluster, such as horizontal pod autoscaling, rolling updates, and log aggregation. These features help to ensure that your applications are running smoothly and efficiently within the cluster.

The Kubernetes network model:

Every Pod in a cluster gets its own unique cluster-wide IP address. This means you do not need to explicitly create links between Pods and you almost never need to deal with mapping container ports to host ports.

This creates a clean, backwards-compatible model where Pods can be treated much like VMs or physical hosts from the perspectives of port allocation, naming, service discovery, load balancing, application configuration, and migration.

Kubernetes imposes the following fundamental requirements on any networking implementation (barring any intentional network segmentation policies):

pods can communicate with all other pods on any other node without NAT

agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node