**Container orchestration tools**

Kubernetes

Docker swarm

Mesos

Cloud factory

**What are the main features of K8?**

Auto Scaling

Self-Healing

High Availability

Scalability

Disaster recovery

**What is Kubernetes?**

Kubernetes is an open source container orchestration tool which is used to automate the deployment, scaling and operation of the container-based applications across the cluster of nodes. It is also designed for managing the services of containerized applications using different methods which provides stability, predictability and high availability.

Or

**Kubernetes is a container orchestration tool like docker swarm, it was initially created by google, but later it was made open source. Using Kubernetes, we can handle all the production related challenges like load balancing, scaling, rolling updates, fail over scenarios etc.**

**The main machine where Kubernetes is configured is called Master and this is controlled by an application called kubeadm.**

**The nodes which take the instructions from master are called as slaves and they use an application called kubeapi.**

**Combination of master and slaves is known as Kubernetes cluster and individual machines in this cluster are called as nodes or minions.**

**Kubernetes objects:**

**POD:**

**This is the smallest object that Kubernetes can work on. With in a pod we have a container, pod is generally termed as a layer of encapsulation on top of container. Kubernetes can not work directly work on containers. Instead, it works on pods. It is the responsibility of the pod to communicate with the container and perform the desired activity.**

**SERVICE:**

**This is an object which works as a router, and it is mainly used for port mapping and network load balancing.**

**NAMESPACE:**

**This is an object used for creating partitions in the Kubernetes cluster.**

**REPLICASET:**

**This is a high-level object which can deploy multiple replicas of pods and it can also be used for scaling.**

**DEPLOYMENT:**

**This is also high-level object which can be used for creating multiple replicas of pods.it can be used for scaling and performing rolling updates.**

**NOTE: Kubernetes can perform container orchestration not only on docker containers, but on containers of any kind.**

**NOTE: kubectl is the program which is responsible for executing Kubernetes commands**

**Architecture of Kubernetes:**

Mainly Kubernetes consists of control plane and worker nodes. Control plane is also known as master node and worker node is also known as slave node.

**Master Node:**

The purpose of the manager node is used to manage the state of the cluster. It is an entry point of all types of administration tasks.

Master node consists of 4 components.

* Kube API Server
* Kube Controller Manager
  + Replication controller
  + Node controller
  + End point controller
  + Service account token controller
* Kube Scheduler
* ETCD
* Cloud Control Manager

**API server** - The API server receives the rest commands sent by the user. It validates the requests, process and executes them. After execution of these commands the result will be stores in ETCD.

**Scheduler** - The scheduler will schedule the tasks to the worker nodes.

**Control Manager** - The controller in the master node preforms a task and manage the state of a cluster. It is also known as cluster. It executes various types of controllers for handling nodes, endpoints etc.

**ETCD** - It is used to store the cluster data. It is a port of master node which is written in GO language.

**Worker Node:**

It is also known as Minions. A worker node is a machine that executes the application using pods. It contains all the essential services which allows a user to assign the resources to the scheduled containers.

Worker node consists of 3 components

Kubelet

Kubeproxy

Container runtime

**Kubelet –** This component is an agent service that executes on each worker node in a cluster. It ensures that the pods and their containers are running smoothly. Every kubelet in each worker node communicates with the master node. It also starts stops and maintain the containers which are organized into pods directly by the master node.

**Kube Proxy** – It is a proxy service of kubernetes. The main aim of this component is request forwarding. Each node interacts with the kubernetes service through kube proxy.

**What is pod?**

Basically, Pods are simply the smallest unit of execution in Kubernetes, consisting of one or more containers.

The lifecycle of a pod is tied to its host node. A pod once created remains in a node until:

1. The pod’s process is terminated.
2. The pod object is deleted.
3. There is a lack of resources for the pod to run on.
4. The host node fails or is terminated.

**What is node?**

The node is the smallest unit of compute hardware in a Kubernetes cluster.

**What is Cluster?**

A Kubernetes cluster is a set of nodes and master that run on containerized applications.

**Methods to install Kubernetes cluster**

Minikube – For learning Purpose with limitations

Kubeadm – We should create cluster manually from the scratch with VPC, Subents, vm’s, Auto Scaling etc. We should be responsible for this cluster

KOPS -

Kubespray

Kubernetes as a service – Managed service like EKS, GKS & AKS

**Basic Commands**

1. kubectl get pods = To list the pods
2. kubectl get pods -o wide = To list the pods with all the details.
3. Kubectl get pods -n kube-system = To see the default pods running while cluster launched
4. kubectl get nodes = To list the nodes
5. kubectl get nodes -o wide = To list the nodes with all details. (O stands for output)
6. kubectl delete pods <pod name> = To delete the particular pod
7. kubectl delete --all pods = To delete all the pods
8. ps = To list the running processes
9. ps aux = To list the process which are running in background also
10. kubectl create -f <pod file name> = To create a pod from definition file
11. kubectl get replicaset = To list the replicas
12. **kubectl replace -f <filename> = To replace the content in a file**
13. kubectl describe node <node name> = To know the full details of the node
14. kubectl describe pods <pod name> = To know the full details of the pod
15. kubectl describe pods <pod name> | less = To know the less details of the pod
16. kubectl exec -it <pod name> --bash = To enter the interactive terminal of pod
17. kubectl exec -it <pod name> -c <container name> --bash = To enter into the interactive terminal of a container if multiple containers are running in a single pod
18. kubectl get all = To list all the replicas along with details
19. kubectl get namespaces = To list the namespaces
20. kubectl get pods -n <namespace> = To list the pods in a particular name space
21. kubectl get pods -n <namespace> -o wide = To list the pods in a particular name space with all the details
22. kubectl get service = To list the services (i.e Load Balancer, Nodeport, Cluster IP)
23. kubectl get pods -n kube-system = To list the pods running when Kubernetes started
24. kubectl run --image <image name> <pod name> = To create a pod with nginx image
25. kubectl run --image <image name> <pod name> --env MYSQL\_ROOT\_PASSWORD = <your password> = To create an mysql server
26. kubectl get nodegroup --cluster=<clustername> = To list the nodes the that particular cluster
27. kubectl delete nodegroup --cluster=<clustername> --name=<nodegroup> = To delete the node group in a cluster
28. kill <container process id> = To kill the pod
29. Kubectl token create –print-join-command = To regenerate the token id in master node
30. Kubectl version –short = To check the version of kubectl
31. Kubectl cluster-info = To check the cluster info

How to create a self-managed Kubernetes cluster?

Step-1

Launch one server with t2.medium server with 10gb storage. Hardware requirements

Step-2

Update the server by using sudo apt-get update.

Install AWS CLI package by using “sudo apt-get install awscli”

Step-3

Create user with administration permissions

Save the Access Key, Secret password etcc

Step-4

Run “aws configure” command to initiate the AWS CLI

Provide Access key

Provide Secret Key

Enter Region in where you want to create a cluster

Step-5

Install kubectl by using the below link

<https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>

After you entered the above page

Select linux

Select the version of Kubernetes which you wanted to install

Run the given links in our command line from a to f to install kubectl

Step-6

Install eksctl by using below link

<https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html>

After you entered the above page

Select linux

Run the given links in our command line form a to c

Step-7

Now we need to create a cluster by using the below command

eksctl create cluster --name=<name of the cluster> \

                                --region=<required region> \

                               --zones=<enter zones with separated by comma> \

                              --without-nodegroup

If we are not mentioning the zones and node group in the above command, AWS will automatically create nodes with high configuration servers.

By using the above command, we created a master in our cluster. To check the same fire this command “eksctl get cluster”

Now run the below command

eksctl utils associate-iam-oidc-provider \

                            --region <selected region in step-7> \

                           --cluster <name given in step-7> \

                           --approve

Step-8

Now we need to create nodes by using the below command

eksctl create nodegroup --cluster=<cluster name> \

                               --region=<selected region> \

                                --name=eksdemo10-ng-public1 \

                               --node-type=t3.medium \

                             --nodes=2 \

                             --nodes-min=2 \

                             --nodes-max=4 \

                            --node-volume-size=20 \

                            --ssh-access \

                            --ssh-public-key=<keypair name> \

                           --managed \

                           --asg-access \

                           --external-dns-access \

                          --full-ecr-access \

                          --appmesh-access \

                          --alb-ingress-access

To check the node, whether it is created or not, fire this command

“kubectl get nodes” – To list the nodes

“kubectl get nodes -o wide” – To list the nodes with all the details

Step-9

Basic commands about cluster and nodes

eksctl get cluster = To list the clusters

eksctl get nodegroup --cluster=<clustername> = To list the nodes in the cluster

kubectl get nodes = To list the nodes

kubectl get nodes -o wide = To list the nodes with full details

**Note: if we terminate a node in the cluster, it will automatically create by the cluster. Hence to delete the nodes and cluster, we should follow some steps.**

Step-10

Deleting nodes and cluster

We need to delete node group created by cluster first, then we should delete cluster.

If we delete the cluster, we won’t be able to delete the node groups.

eksctl delete nodegroup --cluster=<clusterName> --name=<nodegroupName> = To delete the node group

eksctl delete cluster <clusterName> = To delete the cluster

We can install Gcloud in AWS ec2 instance using gcloud SDK kit

Some gcloud commands useful in ec2 instance

Gcloud init = to initialiaze the google cloud

Gcloud configure list = To see the gcloud account details given in ec2 machine

Gcloud compute instance list = To see the gcloud instances are running

Gcloud container clusters list = To see the Kubernetes cluster details in gcloud

Gcloud version = To check the gcloud version

Gcloud components list =

Gcloud components install <package name> = To install package in gcloud

**Definition file objects in kubernetes:**

We have different kinds of objects in Kubernetes. Most commonly used components are

1. Pod
2. Service
3. Replication Controller
4. ReplicaSet
5. Deployment
6. Statefulset
7. Secret
8. Namespace
9. Configmap
10. Volumes
11. Ingress

In each object we have 4 common top-level elements. Those are

1. apiVersion
2. kind
3. metadata
4. spec

**apiVersion:**

When we create Kubernetes resource manifests, one of the first important things that we need to specify for the resource is the apiVersion. For many of the common resources, you might be able to “guess” accurately, but it’s a good skill to be able to figure this out in your cluster. The format of the apiVersion is api\_group/version.

An object definition in Kubernetes requires a apiVersion field. When Kubernetes has a release that updates what is available for you to use—changes something in its API—a new apiVersion is created

we have two api versions, ie v1 and apps/v1. The below are the apiVersions for the different objects as discussed above

1. Pod - v1
2. Service - v1
3. Replication Controller - v1
4. ReplicaSet - apps/v1
5. Deployment - apps/v1
6. Statefulset -
7. Secret -
8. Namespace - v1

**alpha**

API versions with ‘alpha’ in their name are early candidates for new functionality coming into Kubernetes. This is not stable for use. These may contain bugs.

**beta**

‘beta’ in the API version name means that testing has progressed past alpha level and that the feature will eventually be included in Kubernetes. Although the way it works might change, and the way objects are defined may change completely, the feature itself is highly likely to make it into Kubernetes in some form.

**stable**

They are safe to use. These do not contain ‘alpha’ or ‘beta’ in their name.

**v1**

v1 version was the first stable version release of the Kubernetes API. It contains many core objects in Kubernetes.

**apps/v1**

apps is the most common API group in Kubernetes, with many core objects. It includes various functions like running applications on Kubernetes, like Deployments, RollingUpdates, and ReplicaSets.

**autoscaling/v1**

This API version autoscaling/v1 allows pods to be autoscaled based on different resource usage metrics. Kubernetes autoscaling **helps optimize resource usage and costs by automatically scaling a cluster up and down.**Autoscaling is totally dependent on demand.

This stable version includes support for only CPU scaling, but future alpha and beta versions will allow you to scale based on memory usage and custom metrics.

**batch/v1**

The batch API group contains objects related to batch processing and job-like tasks. This apiVersion is the first stable release of these API objects.

**batch/v1beta1**

A beta version release with new functionality for batch objects in Kubernetes, including CronJobs that like you can set a specific time and the job will execute at that time or periodicity.

**certificates.k8s.io/v1beta1**

This API version plays an important role for validate network certificates for secure communication in your cluster.

**extensions/v1beta1**

This version of the API includes many new and commonly used features of Kubernetes. Deployments, DaemonSets, ReplicaSets, and Ingresses all received significant changes in this release. Using extensions/v1beta1 is becoming deprecated—try to use the specific API group where possible, depending on your Kubernetes cluster version.

**policy/v1beta1**

Basically, this apiVersion is used to add the new rules around pod security and the ability to set a pod disruption budget.

**rbac.authorization.k8s.io/v1**

This apiVersion is used for access control. It plays an important role to secure our cluster. It includes more functionality for Kubernetes role-based access control.

**kind:**

It refers to the Kubernetes object. We need to mention the kind while creating a definition object file. (i.e Pod, service, Replication controller, replicaset, deployment, sectret, namespace, statefulset)

**Metadata**:

Additional information about the Kubernetes object like name, labels etc..

**Spec:**

It contains container related info like image name, container name, environment variables, port mapping etc.

**Pod:**

What is pod?

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.

Each pod has its own internal ip address. With that ip addresses pods are communicating with each other. Pods are ephemeral means they can die easily, whenever the new pods gets created it will have its own ip address this will be a problem if we use ip address to expose the application which is running in the pod. To expose the application externally there will be an component

called service and ingress

What is multi container in a pod?

What is pod networking?

What Is inter-pod and intra-pod communication?

What is pod life cycle?

***Pod manifest file***

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: nxinx**

**labels:**

**type: proxy**

**spec:**

**containers:**

* **name: nginx**

**image: nginx:latest**

**ports:**

**- containerPort: 80**

**Replication Controller:**

This is a high-level object used for handling multiple replicas of a specific pod. Here we can perform load balancing and autoscaling. It uses keys like replicas, template etc in the spec section.

**apiVersion: apps/v1**

**kind: ReplicaSet**

**metadata:**

**name: nginx**

**labels:**

**name: any-name**

**type: mention type**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**type: mention type as in above labels**

**template:**

**metadata:**