Introduction to Kubernetes

Official Definition of Kubernetes

Kubernetes is an open-source container orchestration engine designed for automating the deployment, scaling, and management of containerized applications. This open-source project is hosted by the Cloud Native Computing Foundation (CNCF).

Understanding of Kubernetes and Docker

To grasp Kubernetes, also known as K8s, it's essential to have a foundation in Docker. In Docker, we deploy our applications inside containers. However, in Kubernetes, we manage containers on a larger scale, often numbering in the thousands or more, depending on the application's traffic.

Visualizing Docker and Kubernetes

In Docker, imagine a ship containing containers. Now, in Kubernetes, picture that same ship, but this time, it has a steering wheel. Just like a captain operates the ship's wheel to make decisions about its course, Kubernetes acts as the "ship wheel" for managing containers. Kubernetes is an open-source platform, meaning its source code is freely available for anyone to use, modify, and redistribute.

What are Monolithic Architecture and Microservices Architecture?

Monolithic Architecture:

Imagine a restaurant where everything happens in one big kitchen. This kitchen handles taking orders, cooking food, and serving customers all in a single place.

In this scenario, if the kitchen gets too crowded or if there's a problem with one part of the kitchen, it can affect the entire restaurant's operation. If the chef is sick, the entire kitchen may come to a halt, impacting the entire dining experience.

Microservices Architecture: Now, consider a food delivery service like Zomato or Swiggy. Instead of one big kitchen, they have a network of different restaurants, each specializing in a specific type of regional food or cuisine. When you place an order, it's not prepared in a single kitchen rather, each restaurant (microservice) prepares its own portion of the order. These portions are then assembled and delivered to you. If one restaurant has an issue, it doesn't necessarily impact the others. For example, if the burger place is busy, it won't affect the rolls restaurant's ability to fulfill orders.

Key Differences:

- Monolithic architecture is like a single kitchen handling all tasks, while microservices architecture is like multiple specialized restaurants working together.
- Monoliths are typically easier to set up and manage initially, while microservices offer more flexibility and scalability.
- Monoliths can have a single point of failure, while microservices are more fault-tolerant because a failure in one microservice doesn't necessarily affect the others. In the end, Kubernetes helps to achieve microservice-based architecture which is good for business aspects, etc.

Why do we need Kubernetes?

After Docker came into the Picture, the deployment of the applications was very easy on the containers because containers are lightweight. But after some time, there were a lot of issues arose such as managing the huge amount of containers in the Production environment where Containers failed leading to huge Business losses.

After Kubernetes came, it automates many tasks such as:

- Autoscaling of Containers according to the peak or normal hours.
- Load balancing of multiple containers.
- Automatically deployment of containers to the available nodes in the cluster.

● Self-healing if containers fail.

Kubernetes Origins and Open Source:

Kubernetes was created by Google in 2013 in Golang. Initially, Kubernetes was not open source but in 2014, google introduced Kubernetes open source and donated to CNCF.

Languages Supported by Kubernetes

Kubernetes supports both YAML and JSON for configuration.

Features of Kubernetes

● **AutoScaling :** Kubernetes supports two types of autoscaling horizontal and vertical scaling for large-scale production environments which helps to reduce the downtime of the applications.

● **Auto Healing :** Kubernetes supports auto healing which means if the containers fail or are stopped due to any issues, with the help of Kubernetes components(which will talk in upcoming days), containers will automatically repaired or heal and run again properly.

● **Load Balancing :** With the help of load balancing, Kubernetes distributes the traffic between two or more containers.

● **Platform Independent :** Kubernetes can work on any type of infrastructure whether it's On-premises, Virtual Machines, or any Cloud.

● **Fault Tolerance :** Kubernetes helps to notify nodes or pods failures and create new pods or containers as soon as possible

● **Rollback :** You can switch to the previous version.

● Health Monitoring of Containers : Regularly check the health of the monitor and if any container fails, create a new container.

● **Orchestration :** Suppose, three containers are running on different networks (On-premises, Virtual Machines, and On the Cloud). Kubernetes can create one cluster 8 that has all three running containers from different networks.

Alternatives of Kubernetes

● **Docker Swarm**

- **Apache Mesos**
- **Openshift**
- **Nomad, etc**

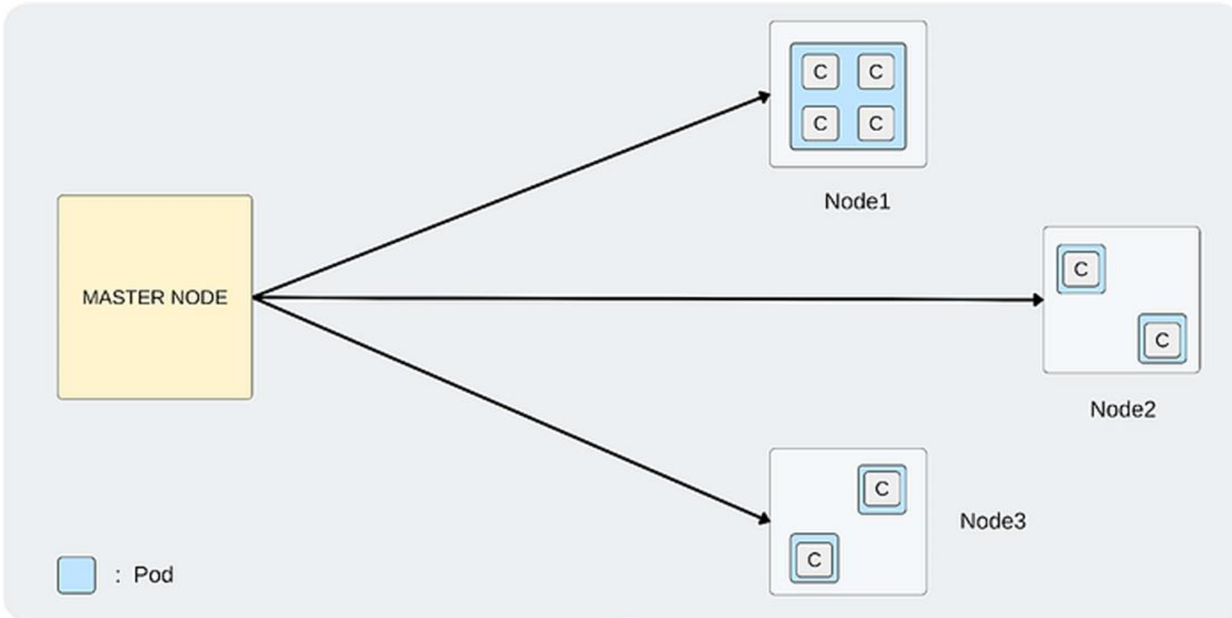
We don’t need to know the other alternative in depth except Docker Swarm as our main focus is Kubernetes.

Difference between Docker Swarm and Kubernetes

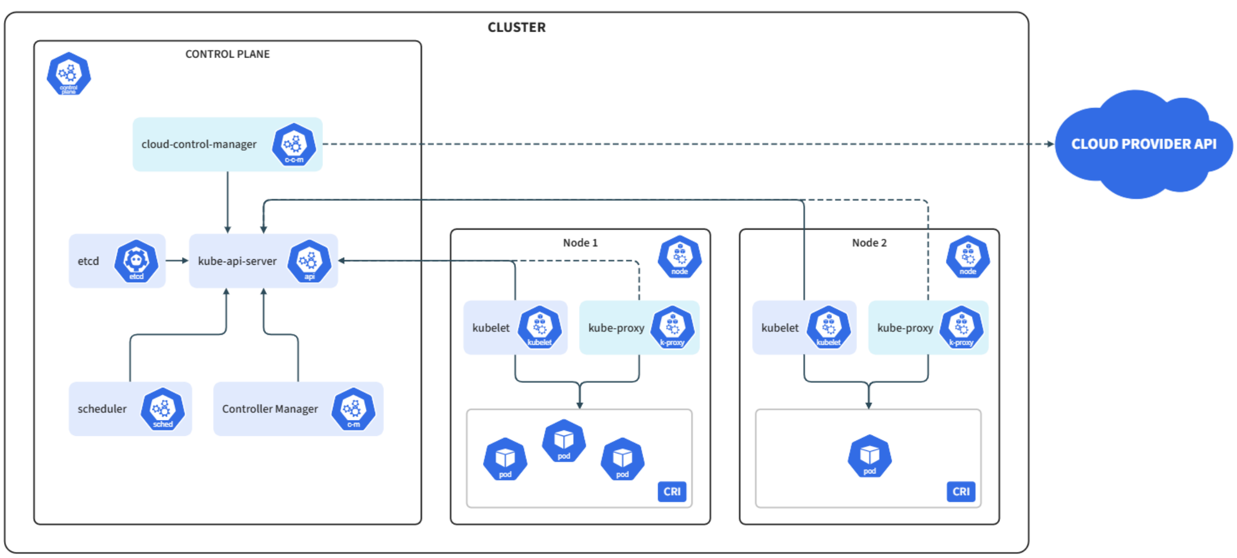
Docker Swarm VS Kubernetes

	DOCKER SWARM	KUBERNETES
INSTALL & CONFIGURATIONS	Quite Easy and Fast	Complicated and Time Consuming
SUPPORTS	Only work with Docker Containers	Can work with any other containers such as Docker, ContainerD, etc.
DATA VOLUMES	Can be shared with any other Containers	Can be shared to the same pod’s containers
GUI	Not Supported	Supported
AUTOSCALING	Not Supported	Supported

Master-Slave/ Client-Server Architecture in Kubernetes



Kubernetes Architecture- Master Node



Kubernetes follows client-server architecture where the Master Node and Worker node exist which constitutes a 'Kubernetes Cluster'. We can have multiple worker nodes and Master nodes according to the requirement.

Control Plane

The control plane components, including the API server, etcd, scheduler, and controller manager, are typically found on the master node(s) of a Kubernetes cluster. These components are responsible for managing and controlling the cluster as a whole .

Master Node

The master node is responsible for the entire Kubernetes cluster and manages all the activities inside the cluster in which master nodes communicate with the worker node to run the applications on the containers smoothly. Master Node has four primary components which help to manage all the things that we have discussed earlier:

1. API Server: In Simple terms, after installing the kubectl on the master node developers run the commands to create pods. So, the command will go to the API Server, and then, the API Server forwards it to that component which will help to create the pods. In other words, the API Server is an entry point for any Kubernetes task where the API Server follows the hierarchical approach to implement the things.

2. Etcd : Etcd is like a database that stores all the pieces of information of the Master node and Worker node(entire cluster) such as Pods IP, Nodes, networking configs, etc. Etcd stored data in key-value pair. The data comes from the API Server to store in etc.

3. Controller Manager: The controller Manager collects the data/information from the API Server of the Kubernetes cluster like the desired state of the cluster and then decides what to do by sending the instructions to the API Server.

4. Scheduler : Once the API Server gathers the information from the Controller Manager, the API Server notifies the Scheduler to perform the respective task such as increasing the number of pods, etc. After getting notified, the Scheduler takes action on the provided work. Let's understand all four components with a real-time example.

Master Node — Mall Management:

• In a shopping mall, you have a management office that takes care of everything. In Kubernetes, this is the Master Node.

• The Master Node manages and coordinates all activities in the cluster, just like mall management ensures the mall runs smoothly.

kube-apiserver — Central Control Desk:

• Think of the kube-apiserver as the central control desk of the mall. It's where all requests (like store openings or customer inquiries) are directed.

• Just like mall management communicates with stores, kube-apiserver communicates with all Kubernetes components.

etcd — Master Records:

• etcd can be compared to the master records of the mall, containing important information like store locations and hours.

• It's a key-value store that stores configuration and cluster state data.

kube-controller-manager — Task Managers:

• Imagine having specialized task managers for different mall departments, like security and maintenance.

• In Kubernetes, the kube-controller-manager handles various tasks, such as ensuring the desired number of Pods are running.

kube-scheduler — Scheduler Manager:

• Think of the kube-scheduler as a manager who decides which employees (Pods) should work where (on which Worker Node).

• It ensures even distribution and efficient resource allocation.

Kubernetes Architecture- Worker Node

Worker Node

The Worker Node is the mediator who manages and takes care of the containers and communicates with the Master Node which gives the instructions to assign the resources to the containers scheduled. A Kubernetes cluster can have multiple worker nodes to scale resources as needed.

The Worker Node contains four components that help to manage containers and communicate with the Master Node:

1. Kubelet : kubelet is the primary component of the Worker Node which manages the Pods and regularly checks whether the pod is running or not. If pods are not working properly, then kubelet creates a new pod and replaces it with the previous one because the failed pod can't be restarted hence, the IP of the pod might be changed. Also, kubelet gets the details related to pods from the API Server which exists on the Master Node.

2. Kube-proxy : kube-proxy contains all the network configuration of the entire cluster such as pod IP, etc. Kube-proxy takes care of the load balancing and routing which comes under networking configuration. Kube-proxy gets the information about pods from the API Server which exists on Master Node.

3. Pods : A pod is a very small unit that contains a container or multiple containers where the application is deployed. Pod has a Public or Private IP range that distributes the proper IP to the containers. It's good to have one container under each pod.

4. Container Engine : To provide the runtime environment to the container, Container Engine is used. In Kubernetes, the Container engine directly interacts with container runtime which is responsible for creating and managing the containers. There are a lot of Container engines present in the market such as CRI-O, containerd, rkt(rocket), etc. But Docker is one of the most used and trusted Container Engine. So, we will use that in our upcoming day while setting up the Kubernetes cluster.

Let's continue to understand all four components with a real-time example. Worker Nodes — Storefronts:

Kubelet — Store Managers:

● In each store (Worker Node), you have a store manager (Kubelet) who ensures employees (Pods) are working correctly.

● Kubelet communicates with the Master Node and manages the Pods within its store.

kube-proxy — Customer Service Desk:

● kube-proxy acts like a customer service desk in each store. It handles customer inquiries (network requests) and directs them to the right employee (Pod).

● It maintains network rules for load balancing and routing.

Container Runtime — Employee Training:

● In each store, you have employees (Pods) who need training to perform their tasks.

● The container runtime (like Docker) provides the necessary training (runtime environment) for the employees (Pods) to execute their tasks.

Setting up Kubernetes cluster on Your Machine

(Click here ^ you will be diverted to github repo)

```
root@master-node:~# kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
master-node.com      Ready    control-plane   19h   v1.30.9
worker1-node.com     Ready    <none>          18h   v1.30.9
worker2-node.com     Ready    <none>          18h   v1.30.9
root@master-node:~#
```

NOTE : - We need to run “`export KUBECONFIG=/etc/kubernetes/admin.conf`” this command every time when we login on cluster. So we will put it into /root/.bashrc file.

We will discuss later that why we need to perform this.

Creating Your First Pod

Lets discuss some basic commands

kubectl run nginx –image=nginx

```
root@master-node:~#  kubectl run nginx --image=nginx
pod/nginx created
root@master-node:~#
root@master-node:~#  kubectl get pods
NAME     READY   STATUS    RESTARTS   AGE
nginx    1/1     Running   0          11s
root@master-node:~#
```

kubectl get pods

kubectl delete pod nginx –force

```
root@master-node:~#  kubectl run nginx --image=nginx
pod/nginx created
root@master-node:~#
root@master-node:~#  kubectl get pods
NAME     READY   STATUS    RESTARTS   AGE
nginx    1/1     Running   0          11s
root@master-node:~#
root@master-node:~#
root@master-node:~#  kubectl delete pod nginx --force
Warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. T
he resource may continue to run on the cluster indefinitely.
pod "nginx" force deleted
root@master-node:~# kubectl get pods
No resources found in default namespace.
root@master-node:~#
```

kubectl get nodes –o wide

```
root@master-node:~# kubectl get nodes -o wide
NAME                STATUS   ROLES           AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
    KERNEL-VERSION   CONTAINER-RUNTIME
master-node.com     Ready    control-plane   19h   v1.30.9   172.31.26.175   <none>        Ubuntu 22.04.5 LT
S   6.8.0-1021-aws   containerd://1.7.25
worker1-node.com    Ready    <none>          19h   v1.30.9   172.31.17.146   <none>        Ubuntu 22.04.5 LT
S   6.8.0-1021-aws   containerd://1.7.25
worker2-node.com    Ready    <none>          19h   v1.30.9   172.31.25.98    <none>        Ubuntu 22.04.5 LT
S   6.8.0-1021-aws   containerd://1.7.25
root@master-node:~#
```

kubectl describe node worker2-node.com

```
root@master-node:~# kubectl describe node worker2-node.com
Name:               worker2-node.com
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker2-node.com
                    kubernetes.io/os=linux
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: unix:///var/run/containerd/containerd.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Mon, 10 Feb 2025 18:37:18 +0000
Taints:             <none>
Unschedulable:      false
Lease:
  HolderIdentity:  worker2-node.com
  AcquireTime:     <unset>
  RenewTime:       Tue, 11 Feb 2025 14:15:30 +0000
Conditions:
  Type                 Status  LastHeartbeatTime                 LastTransitionTime                Reason
                     Message
  ----                 ------  -----------------                 ------------------                ------
                     -------
  NetworkUnavailable   False   Tue, 11 Feb 2025 13:23:19 +0000   Tue, 11 Feb 2025 13:23:19 +0000   WeaveIsU
p                    Weave pod has set this
  MemoryPressure       False   Tue, 11 Feb 2025 14:13:25 +0000   Mon, 10 Feb 2025 18:37:18 +0000   KubeletH
asSufficientMemory   kubelet has sufficient memory available
  DiskPressure         False   Tue, 11 Feb 2025 14:13:25 +0000   Mon, 10 Feb 2025 18:37:18 +0000   KubeletH
asNoDiskPressure     kubelet has no disk pressure
  PIDPressure          False   Tue, 11 Feb 2025 14:13:25 +0000   Mon, 10 Feb 2025 18:37:18 +0000   KubeletH
asSufficientPID      kubelet has sufficient PID available
  Ready                True    Tue, 11 Feb 2025 14:13:25 +0000   Mon, 10 Feb 2025 18:39:51 +0000   KubeletR
eady                 kubelet is posting ready status
```

How Enable kubectl bash-completion

To enable kubectl bash-completion in Kubernetes, run the following command:

```
source <(kubectl completion bash)
```

To make it persistent across sessions, add it to your `.bashrc` or `.bash_profile`:

```
echo 'source <(kubectl completion bash)' >> ~/.bashrc
source ~/.bashrc
```

For better CLI experience, you can also enable kubectl autocompletion with kubectl alias:

```
echo 'alias k=kubectl' >> ~/.bashrc
echo 'complete -o default -F __start_kubectl k' >> ~/.bashrc
source ~/.bashrc
```