Kubernetes:
For orchestration, always Kubernetes is better than Docker swarm

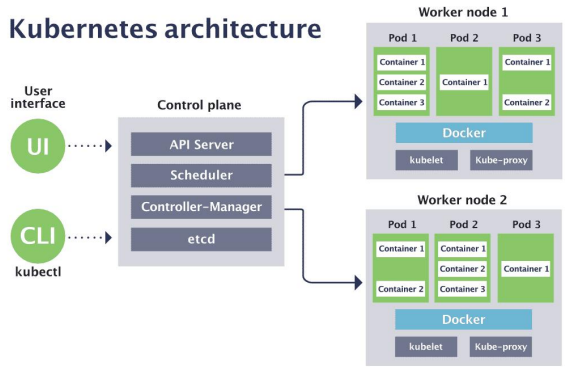
It is free and open-source --> Developed by Google --> GO programming language is used to develop Kubernetes.
Kubernetes is an Orchestration platform --> Used to manage containers (create, start, stop, delete, scale-up, scale-down containers)
It provides framework for managing the complex task of deploying, scaling and operating applications in containers

Advantages:
1. Self-healing: if any container gets crashed, it will be automatically replaced with a new container immediately
2. Auto-scaling: Based on demand, containers count will be increased or decreased
3. Load-balancing: Load will be distributed to all containers equally, which are up and running

Docker vs Kubernetes:
What's the purpose of Docker?
It is for containerization, to containerize the application. Containerization platform. It is for packaging our application code and dependencies as a single unit for the execution is referred as Containerization.
What's the significance of Kubernetes?
It is an orchestration platform. It is for the orchestration purpose. Managing the containers that got created.
https://kubernetes.io/docs/concepts/architecture/

Kubernetes Architecture
--> K8s follows cluster architecture
--> Cluster refers to group of servers (machines, VMs)
--> in K8s cluster, we will have a Control node (Master node) and Worker nodes

K8s Cluster Components:
1. Control Node (Master Node)
    a) In the control node, we have something called as an API server
    b) Scheduler
    c) Controller-Manager
    d) etcd

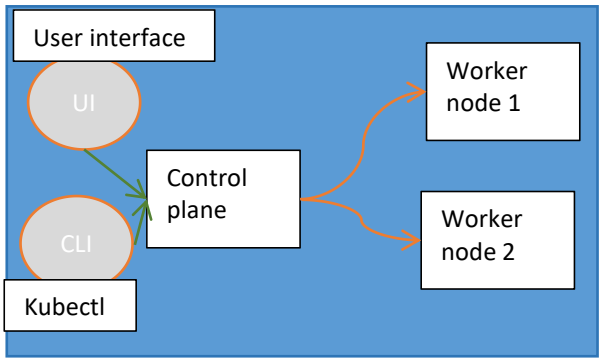


2. Worker Node
    a) Kubelet
    b) Kube proxy
    c) Docker engine
    d) POD

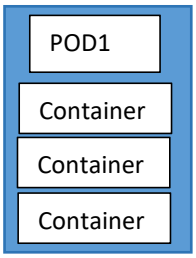
e) Within the POD, we have the Container

Kubernetes Cluster Architecture

Two important parts in Kubernetes Architecture



In Control plane we have 4 components: API server, Scheduler, Controller-manager, etcd
In every Worker node, we have kubelet, kube-proxy, docker, pods (Pod1 will have many containers)



--> To deploy our application using K8s we need to communicate with Control plane (Master node)
--> We usually use KUBECTL (CLI) to communicate with Control plane
--> API server will receive the request given by kubectl and it will store the request with pending status in ETCD
--> ETCD is an internal database of k8s cluster.
--> any pending requests in ETCD will be identified by Scheduler then will schedule tasks in Worker node. Scheduler will identify the Worker node to schedule this pending request with the help of Kubelet. Kubelet is a Node agent, it will maintain information about all Worker node.
Scheduler will go to ETCD, identify pending requests, then it will schedule tasks by identifying the Worker node. Kube proxy provides network for Cluster communication. Controller-manager is used to verify all the tasks are working as per expectations or not. Docker engine will be present in the Worker node. In K8s architecture, will container be directly created under worker node? Containers will be created inside the Pod. All containers will be there inside the Pod only.
--> Scheduler will identify the pending request in ETCD and it will identify Worker node to schedule the task
--> Scheduler will identify Worker node using Kubelet
--> Kubelet is a Node agent, which will maintain all the worker node information
--> Kube proxy will provide network for Cluster communication
--> Controller-manager will verify all the tasks, which have been assigned are working fine as expected or not
--> Docker engine would be present in the Worker node to run Docker container
--> In K8s, Containers will be created inside POD --> POD is the smallest building block that we could create in a K8s cluster
--> Generally in K8s, everything is represented as POD only
--> Note: we don't directly work with containers they stay within Pods

POD:
POD is the smallest building block in the K8s cluster and applications will be deployed as a Pod in K8s.
We can create multiple Pods for one application.
In order to create a Pod, we use a Yaml file (Manifest YML) and in Pod manifest YML we will configure our Docker image
If a Pod is damaged/deleted/crashed, then K8s will create a new Pod (Self-healing).
If an application is running in multiple Pods then K8s will distribute the load to all the running Pods. This is the concept of Load balancers.
Pods could be increased or decreased automatically based on load (Scalability)

K8s Cluster Setup:
1. Mini Kube --> Single node cluster --> Only for practice
2. Kubeadm cluster --> Self-managed cluster (everything is managed by us only). we are responsible for everything. We are going to create machines, control node etc
3. Provider Managed Cluster --> Ready made cluster --> Provider will take care of everything.
Examples: AWS EKS, Azure, AKS, GCP GKE etc.
Note: Provider-managed clusters they are paid they are chargeable

Practical steps for Kubernetes cluster setup
Step 1: Create EKS management host in AWS
Launch a Linux machine (Ubuntu VM) using AWS EC2 (t2.micro)
Connect to this machine and install Kubectl
Install Kubectl
curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/

sudo apt update && sudo apt install -y unzip

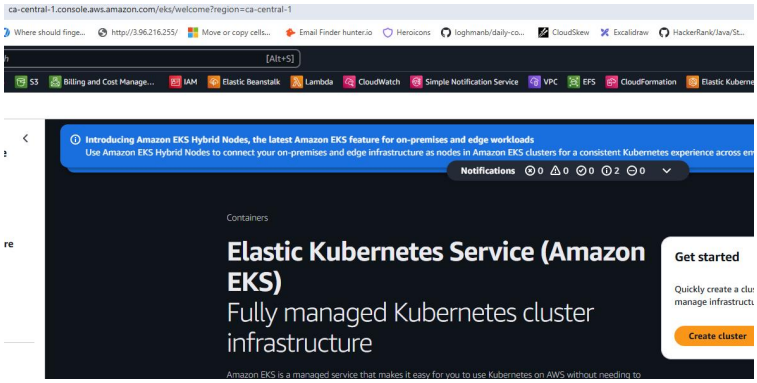
Install awscli
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

Cleanup
rm -rf awscliv2.zip

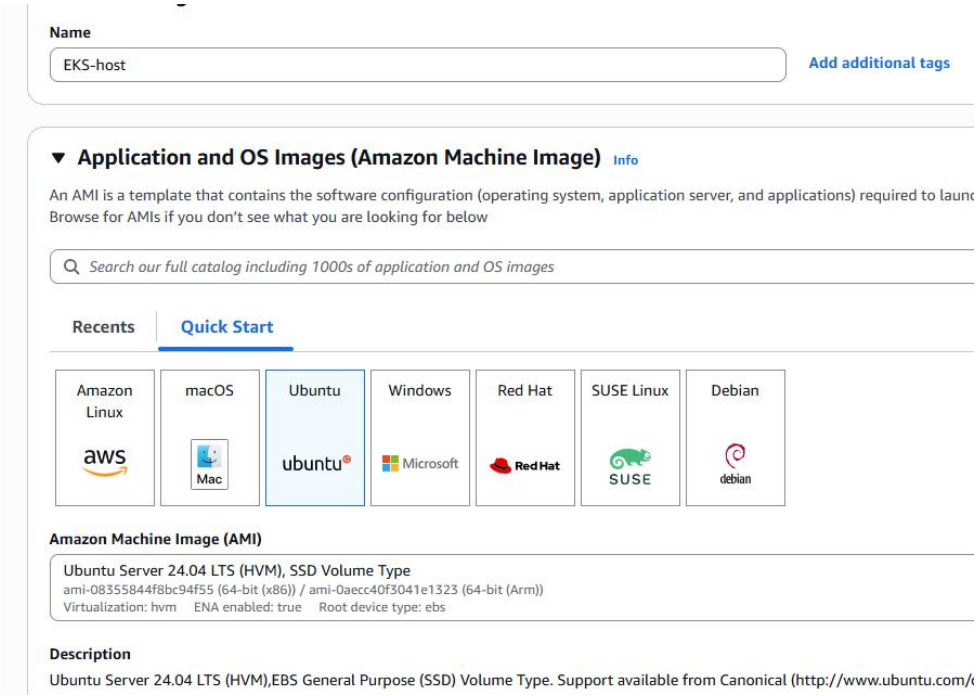
Verify installation
aws --version

Install eksctl
curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
tar -xzf eksctl.tar.gz
sudo mv eksctl /usr/local/bin/
eksctl version

Create AWS IAM role and attach to EC2 host



No option to temporarily stop the cluster you have to delete



curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   138  100   138    0     0   2031      0 --:--:-- --:--:-- --:--:--  2029
100 57.3M  100 57.3M    0     0  2259k      0  0:00:25  0:00:25 --:--:-- 2275k
ubuntu@ip-172-31-9-165:~$
```

```
ubuntu@ip-172-31-9-165:~$ kubectl version
Client Version: v1.33.1
Kustomize Version: v5.6.0
The connection to the server localhost:8080 was refused - did you specify the right host or port?
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ sudo apt update && sudo apt install -y unzip

```
ubuntu@ip-172-31-9-165:~$ sudo apt update && sudo apt install -y unzip
Hit:1 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:5 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:6 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:7 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [838 kB]
Get:8 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:9 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:10 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:11 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:12 http://ca-central-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
```

ubuntu@ip-172-31-9-165:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 67.2M  100 67.2M    0     0   103M      0 --:--:-- --:--:-- --:--:--  102M
Archive:  awscliv2.zip
   creating: aws/
```

```
ubuntu@ip-172-31-9-165:~$ rm -rf awscliv2.zip
ubuntu@ip-172-31-9-165:~$ ls -l
total 4
drwxr-xr-x 3 ubuntu ubuntu 4096 May 16 18:46 aws
ubuntu@ip-172-31-9-165:~$
```

ubuntu@ip-172-31-9-165:~$ aws --version
aws-cli/2.27.17 Python/3.13.3 Linux/6.8.0-1024-aws exe/x86_64.ubuntu.24

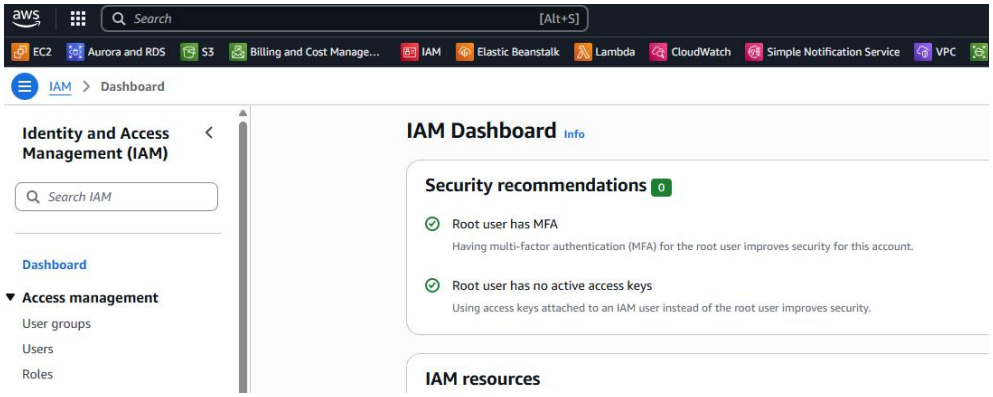
Verify installation

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ aws --version
aws-cli/2.27.17 Python/3.13.3 Linux/6.8.0-1024-aws exe/x86_64.ubuntu.24
ubuntu@ip-172-31-9-165:~$
```

Install eksctl
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/
ubuntu@ip-172-31-9-165:~$ eksctl version
0.208.0

```
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/
ubuntu@ip-172-31-9-165:~$ eksctl version
0.208.0
```

```
ubuntu@ip-172-31-9-165:~$
ubuntu@ip-172-31-9-165:~$ curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" -o eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ tar -xzf eksctl.tar.gz
ubuntu@ip-172-31-9-165:~$ sudo mv eksctl /usr/local/bin/
ubuntu@ip-172-31-9-165:~$ eksctl version
0.208.0
```



PODS
Services (ClusterIP, NodePort, LoadBalancer)
Namespaces
ReplicationController (RS)
ReplicaSet
DaemonSet
StetefulSet
IngressController
HPA
HelmCharts
K8s monitoring (Grafana, Promethues)
EFK stack group setup to monitor app logs

Note: We need a machine where we install kubectl
In the same kubectl Host machine, we need kubectl, awscli and eks cli