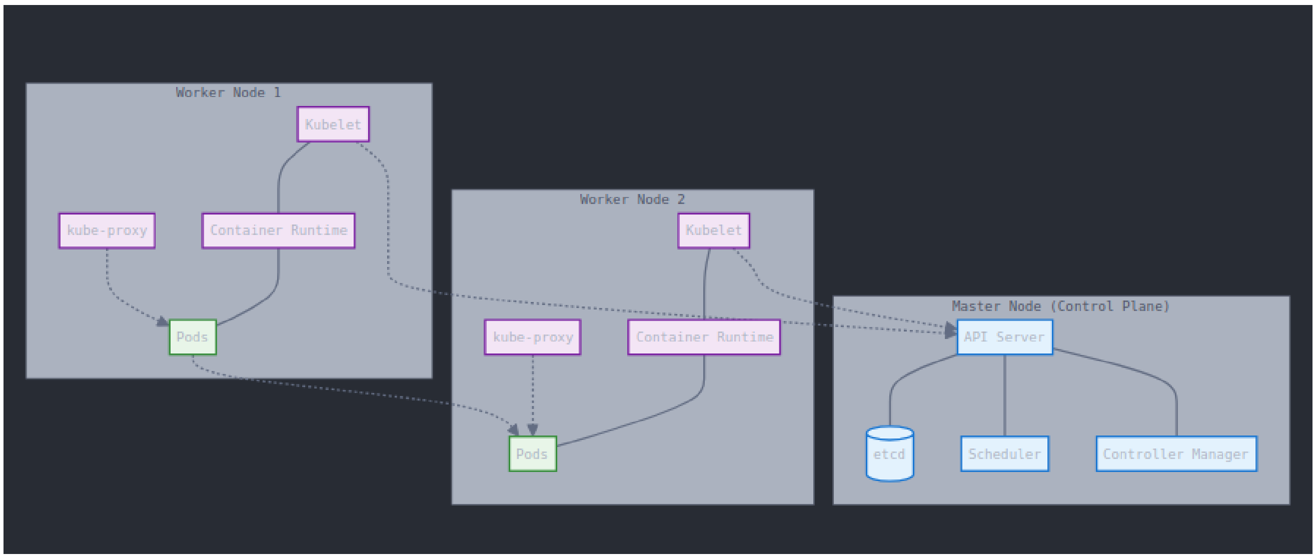
PROJECT Name–Kubernetes
NAME-AASIF MOHD

Why Kubernetes (a group of connected machines called nodes)?
To run applications at the enterprise level reliably and efficiently.

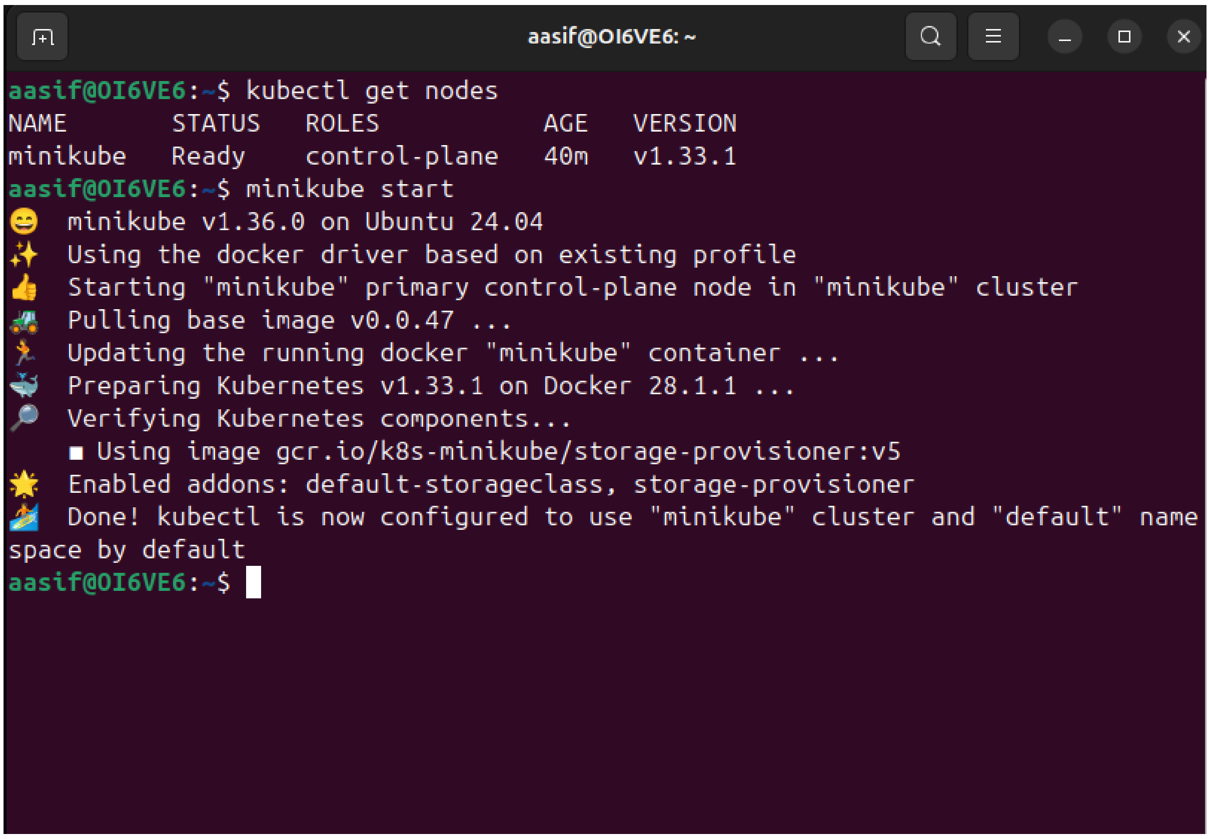
1. **Auto Scaling** – If the traffic or workload increases, Kubernetes will automatically create more containers to handle it.
2. **Auto Healing** – If a container stops working or crashes, Kubernetes will detect it and create a new one to replace it.
3. **Node Failure Handling** – If one machine (node) has a problem, Kubernetes can move the container to another healthy machine.
4. **Why not just Docker?** – Docker alone is useful but doesn't have advanced features like auto-scaling or self-healing. That's why, in large companies, Kubernetes is used along with Docker to manage containers better.

Kubernetes Architecture:

1. **Cluster** – A group of machines (nodes) that run your applications.
2. **Master Node (Control Plane)** – Makes decisions about the cluster (like scheduling and managing the state). Key components:
   a. **API Server** – Entry point to the cluster.
   b. **Scheduler** – Decides where to run new pods.
   c. **Controller Manager** – Handles tasks like restarting failed pods.
   d. **etcd** – Stores all cluster data (like a database).
3. **Worker Nodes** – Where the actual applications (pods/containers) run. Key components:
   a. **Kubelet** – Communicates with the master and runs pods.
   b. **Kube Proxy** – Manages networking and load balancing.
   c. **Container Runtime** – Runs the containers (like Docker or containerd).
4. **Pods** – The smallest unit in Kubernetes, holding one or more containers.



I am Setting up minikube for Learning--



```
aasif@OI6VE6:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   40m   v1.33.1
aasif@OI6VE6:~$ minikube start
😄  minikube v1.36.0 on Ubuntu 24.04
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.47 ...
🏃  Updating the running docker "minikube" container ...
🐳  Preparing Kubernetes v1.33.1 on Docker 28.1.1 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" name
space by default
aasif@OI6VE6:~$
```

I have installed minikube and kubectl

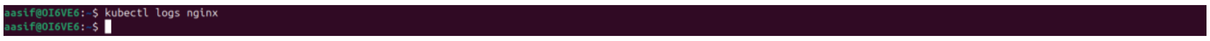
We can debug the pod and check its information by Using the cammands

kubectl describe pod nginx

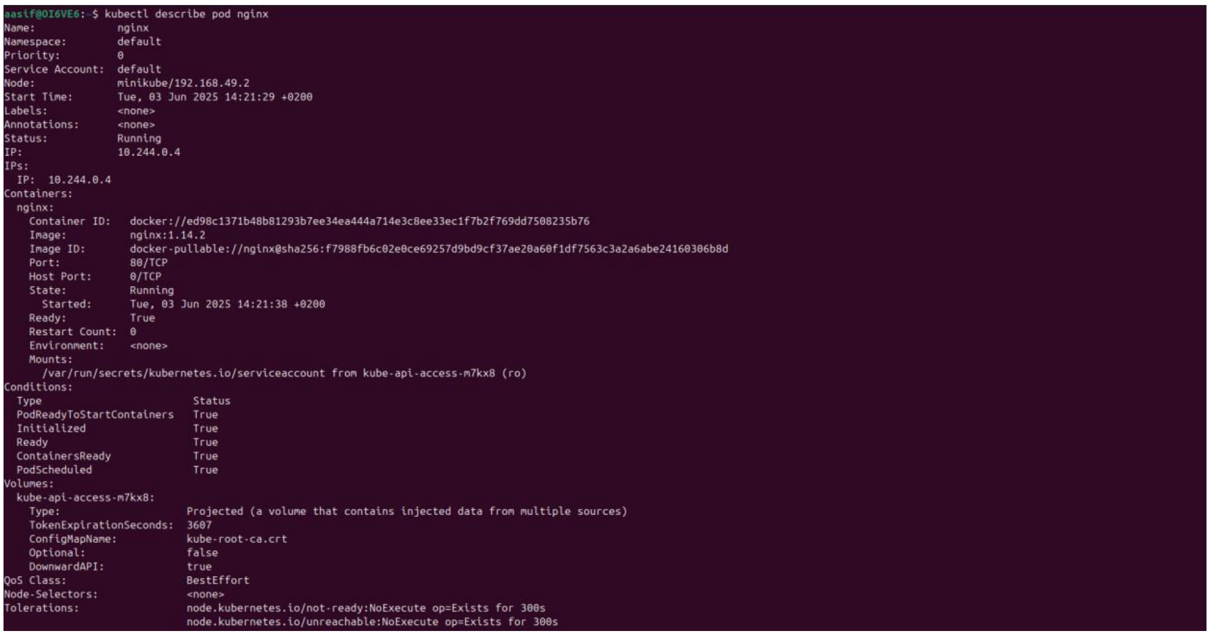
kubectl logs nginx

Kubectl get all - it will show all the things that are running

if there is no log its show nothing



```
aasif@OI6VE6:~$ kubectl logs nginx
aasif@OI6VE6:~$
```



```
aasif@OI6VE6:~$ kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Tue, 03 Jun 2025 14:21:29 +0200
Labels:           <none>
Annotations:      <none>
Status:           Running
IP:               10.244.0.4
IPs:
  IP:  10.244.0.4
Containers:
  nginx:
    Container ID:   docker://ed98c1371b48b81293b7ee34ea444a714e3c8ee33ec1f7b2f769dd7508235b76
    Image:          nginx:1.14.2
    Image ID:       docker-pullable://nginx@sha256:f7988fb6c02e0ce69257d9bd9cf37ae20a60f1df7563c3a2a6abe24160306b8d
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 03 Jun 2025 14:21:38 +0200
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-m7kx8 (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-m7kx8:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    Optional:                false
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
```

Container vs pod vs Deployment-

- **Container**: The actual app in a box.
- **Pod**: A logical unit that wraps one or more containers.
- **Deployment**: The boss that creates and manages multiple Pods for scalability,healing and stability.

REPLICA SET-
When we create a **Deployment** in Kubernetes:

- It **automatically creates** a **ReplicaSet** under the hood.
- That ReplicaSet is responsible for maintaining the **desired number of Pod replicas**.
- If a Pod **crashes**, **gets deleted**, or **fails**, the ReplicaSet will:
  - **Detect it** (via the kubelet and controller manager)
  - **Automatically recreate** a new Pod to match the desired count

KUBERNETES SERVICE- A **Service** in Kubernetes is a way to give a **fixed name and IP address** to a group of Pods so other apps can find and talk to them easily — even if the Pods come and go or change their IPs. It also helps to **share the traffic** between those Pods (load balancing).

Label in kubernetes service- A **label** is a tag added to Pods, and a **Service uses a**

selector to match those labels. This tells the Service which Pods it should send traffic to.

- **Services do not track Pod IP addresses directly. Instead, they use labels to find which Pods to send traffic to.**
- Kubernetes takes care of mapping those labels to the current IPs of the matching Pods **automatically** behind the scenes.

Service Types-

- **ClusterIP** services do not provide external access; they make the app accessible only within the Kubernetes cluster.
- **NodePort** services expose the app on a static port on each node's IP address, allowing external access through those node IPs and ports.
- **LoadBalancer** services create an external load balancer (usually in a cloud environment) that publicly exposes the app to the internet.
- **ExternalName** services do not provide direct access; instead, they map the service to an external DNS name outside the cluster.

Question1-
Difference Between Docker and kubernetes?
Docker is a container platform where kubernetes is a container orchestration enivironment that offers capabilities like auto healing ,auto scaling,clusturing and enterprise level suuport like load balancing.

Q2- Main components of kubernetes architecture-

Control Plane components:

- API Server
- etcd
- Scheduler
- Controller Manager

Worker Node components:

- kubelet
- kube-proxy
- Container Runtime (e.g., containerd, CRI-O)

Q3 main difference between docker swarm and kubernetes?
Kubernetes is better suited for large organisation as it offers more scalibility networking capabilities like policies and huge third party ecosytem suuport

Q4 main difference between docker containerand kubernetes pod?

- **Docker container**: Runs **one container**, managed individually.
- **Kubernetes pod**: Can run **multiple containers together** as a single unit, managed by Kubernetes.

Q5 Namespace in kubernetes-Kubernetes cluster shared by two teams:

Team A works in namespace team-a

Team B works in namespace team-b

Both can use the cluster without interfering with each other's resources.

Q6 what is Kube proxy?- **Kube-proxy** is a network component in Kubernetes that **routes and forwards traffic** to the right pods inside the cluster, enabling communication between services and pods.

Q7 What are the diff types of services within kubernetes

- Cluster IP,Node port Mode,Load Balancer Mode,**ExternalName**

Q8 Difference between nodeport and load balancer-
NodePort exposes a Kubernetes service on a fixed port on every node's IP address, allowing you to access the service using NodeIP:NodePort. It works without a cloud provider but requires manual management of external traffic.

LoadBalancer, on the other hand, automatically provisions an external load balancer (usually from a cloud provider) with a public IP address that distributes traffic to the service, making it easier to access and scale without manual setup.

Q9 what is the role of kubelet?
The **kubelet** is an agent that runs on every Kubernetes node. Its main job is to **make sure containers are running inside pods** as specified by the Kubernetes control plane. It communicates with the master node, manages pod lifecycle, and reports the node's status back to the cluster.
Q10 day to day Activities on kubernets
On a typical day, you deploy applications using YAML files or Helm charts, and regularly check the status of pods, nodes, and services. You update or roll out new versions of your apps, debug issues by viewing logs, and scale applications up or down as needed.

You organize resources using namespaces, monitor the health and resource usage of the cluster, and configure networking and storage. Additionally, you manage security policies, back up important data, and occasionally upgrade the Kubernetes cluster components.

Ingress-
Ingress in Kubernetes is like a **gate** or **reception desk** for your application.

Here's why we use it:

1. **Control traffic**: It helps manage and route external traffic (from the internet) to the right service inside your Kubernetes cluster.
2. **Easy access**: Instead of opening many doors (ports) for every service, Ingress gives one single point of entry.
3. **Smart routing**: You can say, "If someone goes to /shop, send them to the shopping app; if they go to /blog, send them to the blog app."
4. **Security**: It can handle HTTPS (secure connections) for your services, keeping data safe.
5. **Clean setup**: It avoids creating separate LoadBalancers or NodePorts for each app, saving resources and simplifying things.

Example:
Imagine your website has a store, blog, and login system. Ingress helps decide which request goes where, using one URL like www.example.com.

Config maps and Secrets-------

A **ConfigMap** is used to store **configuration data** for your applications in **key-value** format.
This is useful when you don’t want to hardcode settings (like file paths, environment names, etc.) in your app.

Example:

- You have an app that needs the name of the database to connect to.

A **Secret** is like a ConfigMap, but it’s used to store **sensitive data**, like:

- Passwords
- API keys
- Tokens

base64 encoded

RBAC stands for **Role-Based Access Control**.

It’s a way to control **who can do what** in a Kubernetes cluster.

🔁 What happens when you **change a value**?

Environment Variable (env)

- If you change it (like in a ConfigMap or Deployment),
  ➜ **You must restart the pod** to see the new value.
- Reason: Environment variables are **only read when the container starts**.

Volume Mount (volumeMount)

- If you mount a file from a **ConfigMap or Secret**,
  ➜ The container can **see updates automatically** (within ~1–2 minutes).
- No need to restart the pod.

*Use **volume mounts** when:*

- You want to give files to your container.
- Your app needs to read from a file or folder.
- You are sharing files between containers.
- You need to save data (like in a database).

*Use **environment variables** when:*

- You want to pass simple values like text or numbers.
- Your app reads settings from environment variables.
- You don't need to save files or data.

Q What happens when you **change a value**?

When you change an environment variable, the pod must be restarted to pick up the new value.
To avoid this, use a **volume mount** with a ConfigMap or Secret, which updates automatically without a restart.

Prometheus is a **monitoring tool**.
It watches your **Kubernetes cluster** (apps, containers, nodes) and collects **metrics** like:

- CPU usage
- Memory usage
- Network traffic
- How many requests your app is getting

It stores this data so you can check the **health and performance** of your system.

What is Grafana?

Grafana is a **visualization tool**.
It takes the data collected by Prometheus and shows it on **dashboards** using **graphs, charts, and alerts**.

So instead of reading numbers, you can **see how your system is doing** at a glance.

◇ What is a **Custom Resource (CR)**?

A **Custom Resource** is like creating your own type of object in Kubernetes.

Kubernetes already has built-in types like:

- **Pod**
- **Service**
- **Deployment**

But sometimes you want to create something special for your app, like a:

- **"Database"**
- **"Backup"**
- **"Website"**

That's where **Custom Resources** come in — you make your own object with your own rules.

◇ What is a **Custom Resource Definition (CRD)**?

A **CRD** is how you tell Kubernetes about your new type.

Think of it like:

"Hey Kubernetes, I want to create a new kind of object called Database, and it should have fields like engine, version, and size."

Once you create the CRD, you can start making Database objects, just like you make Pods or Services.