Kubernates



Kubernetes follows a **master-worker (control plane-node)** architecture. It manages containerized applications across a cluster of machines (nodes), ensuring high availability, scalability, and reliability.

### 1. Control Plane (Master Node) The control plane makes global decisions about the cluster (e.g., scheduling), and detects and responds to cluster events.

What was the problem to manage the docker container?

* Bhut saare containers ko manage kaun krega
* Autoscaling kaise hoga
* Load Balancing kaise hoga
* Node mar gya to
* Node restart ho gya to

#### Components:

### kube-apiserver

* **Acts as the**front-end (REST API server)**of the cluster.**
* **All commands (e.g., from kubectl) interact here.**
* **Validates and processes requests, talks to etcd, and updates cluster state.**
* **etcd**
  + Key-value store**to hold the entire state of the cluster.**
  + **Stores configuration data, secrets, status, etc.**
  + **Acts as Kubernetes'**database**.**

### kube-scheduler

* + **Assigns pods to nodes based on:**
    - **Resource requirements**
    - **Affinity/anti-affinity rules**
    - **Taints and tolerations**
    - **Node selectors and constraints**

### kube-controller-manager

* + **Runs multiple controllers (loops that manage cluster state):**
    - **Node Controller – Checks node health.**
    - **Replica Set Controller – Maintains desired number of pods.**
    - **Endpoint Controller – Updates services and endpoints.**
    - **Job Controller – Manages batch jobs.**

### cloud-controller-manager *(optional, for cloud environments)*

* + **Handles cloud provider-specific operations like:**
    - **Load balancer provisioning**
    - **Node management**
    - **Volume attachment**

## **⚙️**Node Components (Worker Nodes)

**Nodes are machines (VMs or physical) where application**containers run**.**

#### Components:

### kubelet

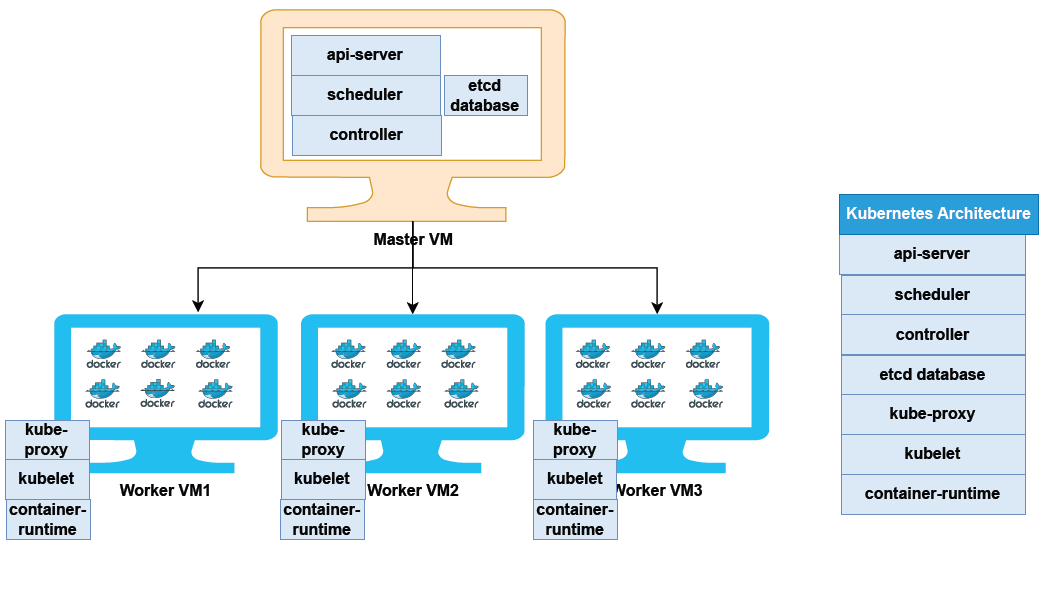
* + **Agent running on each node.**
  + **Ensures containers are running as per PodSpecs.**
  + **Communicates with the API server.**

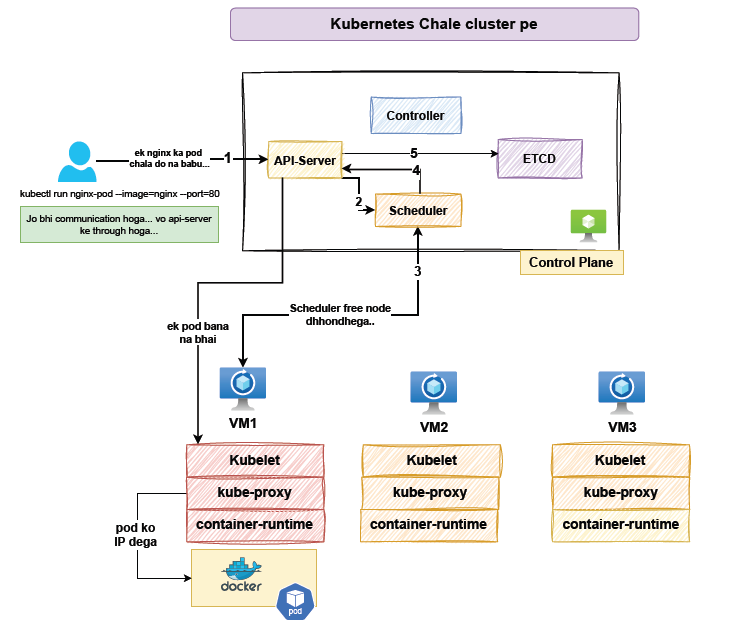
### kube-proxy

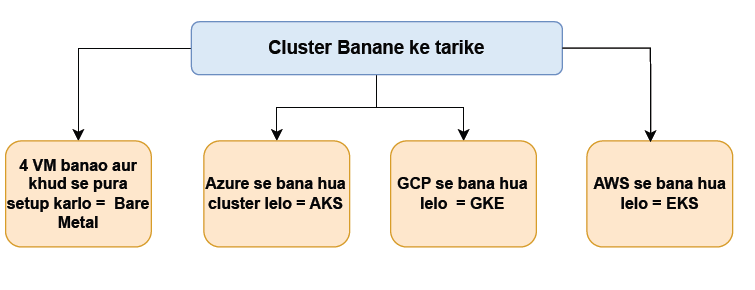
* + **Manages network routing on the node.**
  + **Implements**service abstraction**using iptables or IPVS.**
  + **Handles traffic forwarding to the appropriate pod.**

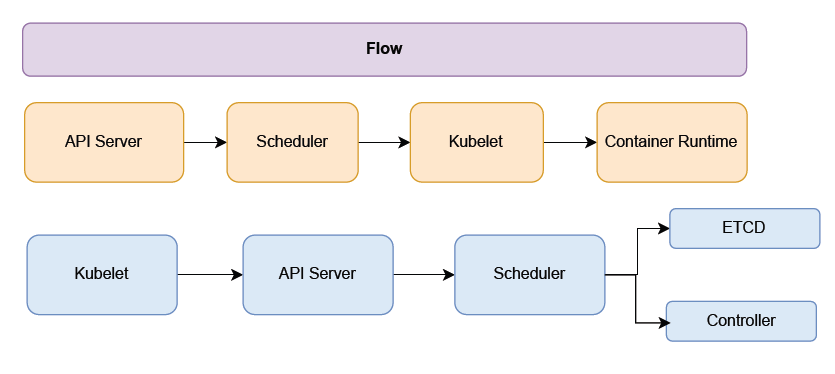
### Container Runtime

* + **Software responsible for**running containers**.**
  + **Examples:**contained**,**Docker**,**CRI-O**.**









Azure Kubernetes Service (AKS) - Notes

🔹 What is AKS?

* AKS is a managed Kubernetes service provided by Microsoft Azure.
* It simplifies deploying, managing, and operating Kubernetes (K8s) clusters in Azure.
* Azure handles control plane (master node) for you (no cost), and you manage worker nodes (agent nodes).

🔹 AKS Architecture

1. Control Plane (Managed by Azure):
   * Manages API server, scheduler, controller manager, etc.
   * Azure hosts and maintains high availability for the control plane.
2. Node Pools (Managed by User):
   * VM Scale Sets hosting the actual Kubernetes worker nodes.
   * You can create multiple node pools for different workloads (e.g., Linux and Windows).
3. Kubernetes API Server Endpoint:
   * Public or private (integrated with Azure VNet).

Resource Type:

* Node = Worker Node: Jo apne cluster se connected hai
* POD = Jiame container chalega

# 📘 Kubernetes Pod YAML & kubectl explain

🚀 What is a Pod in Kubernetes?

* A Pod is the smallest deployable unit in Kubernetes.
* It can contain one or more containers that share the same network and storage.
* Think of a Pod as a wrapper around your application container(s).

🧾 What is a Pod YAML?

Kubernetes resources like Pods are defined in YAML files, which describe what to create and how.

A minimal Pod YAML looks like this:

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- name: mycontainer

image: nginx

🔍 What is kubectl explain?

kubectl explain is a CLI helper that describes the structure and meaning of Kubernetes resource fields. It is useful when you're writing YAML and are not sure what fields to use or how to nest them.

🧠 Basic kubectl explain Commands

| Command | Purpose |
| --- | --- |
| kubectl explain pod | Explains top-level structure of Pod |
| kubectl explain pod.metadata | Shows available metadata fields |
| kubectl explain pod.spec | Describes the Pod spec |
| kubectl explain pod.spec.containers | Shows required fields for containers |
| kubectl explain pod.spec.containers.image | Explains the image field |

Each command shows:

* Type (e.g., string, object, list)
* Description
* Whether it's optional or required

🛠️ How to Build Pod YAML Using kubectl explain

1. Start with:

kubectl explain pod

1. Explore fields like:
2. kubectl explain pod.metadata
3. kubectl explain pod.spec

kubectl explain pod.spec.containers

1. Use this information to build a YAML step-by-step.

For example:

* From spec.containers, you learn name, image are required.
* From metadata, you add a name for the pod.

🔁 Full Tree with --recursive

To get the full hierarchy of the Pod object, use:

kubectl explain pod --recursive

This shows all nested fields in a tree format.

📂 Save Output to a File

To save this full field structure to a file:

kubectl explain pod --recursive > pod\_document.txt

✅ Now you have a reference document to guide your YAML writing.

📄 Example of pod\_document.txt Output (Trimmed)

KIND: Pod

VERSION: v1

FIELDS:

apiVersion <string>

kind <string>

metadata <Object>

name <string>

labels <map[string]string>

spec <Object>

containers <[]Object>

name <string>

image <string>

ports <[]Object>

containerPort <integer>

✅ Final Example: Complete Pod YAML

Based on what we learned, here’s a working Pod YAML:

apiVersion: v1

kind: Pod

metadata:

name: my-nginx-pod

labels:

app: web

spec:

containers:

- name: nginx-container

image: nginx

ports:

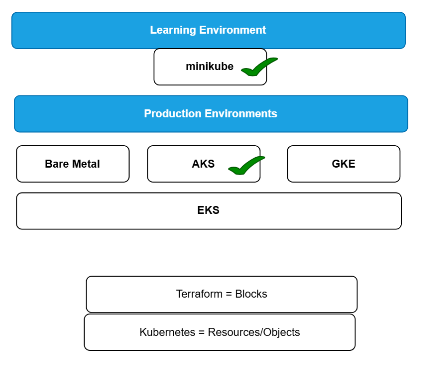
- containerPort: 80

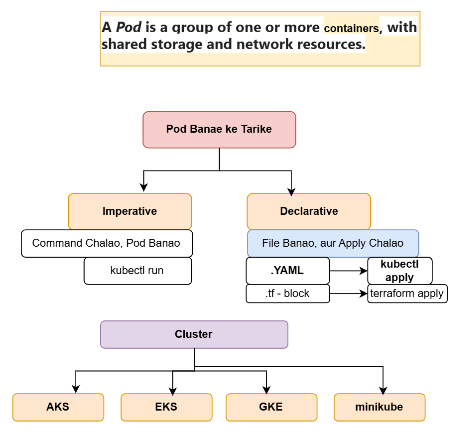
You can apply this with:

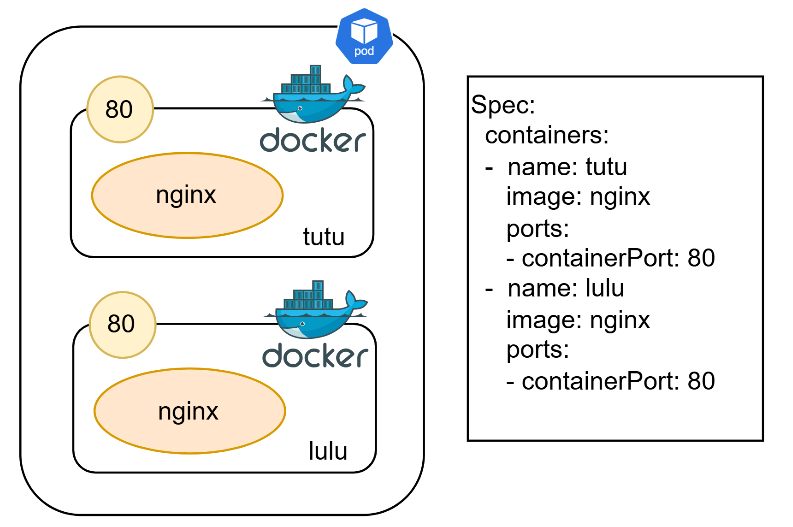
kubectl apply -f mypod.yaml

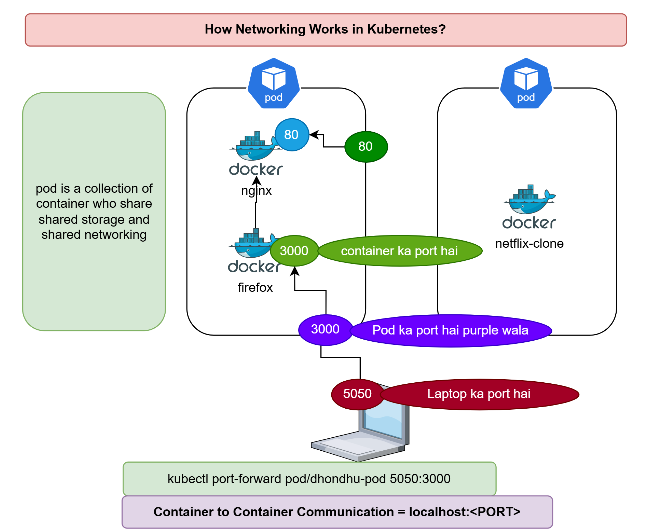
📌 Summary

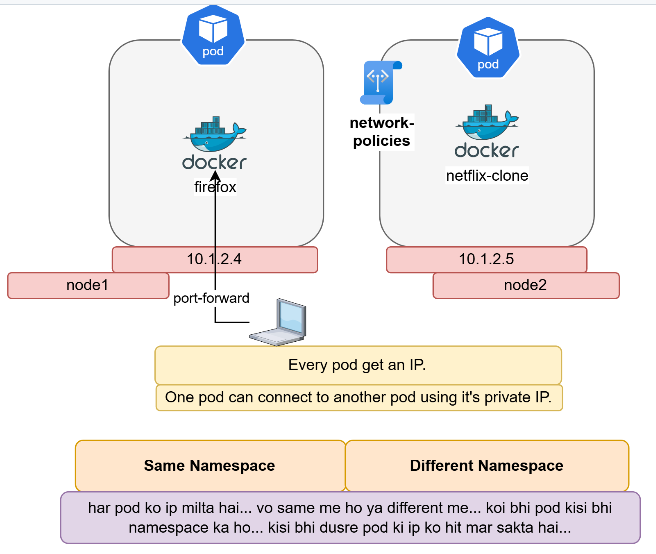
| Topic | Summary |
| --- | --- |
| kubectl explain | Explore Kubernetes object fields |
| --recursive | Show all nested fields in tree format |
| > pod\_document.txt | Save output for offline reference |
| YAML Creation | Use kubectl explain to guide YAML creation without memorizing fields |
| Best Use | When writing YAMLs manually or troubleshooting structure errors |











How to write POD YAML file ?

1. We need to start minikube first inside the local computer on vs code terminal
2. Kubelet get pod >> No pod will be shown here because no POD is available till now
3. kubectl explain pod --recursive > demo.txt >> to download the doc file of pod
4. with the help of demo.txt write the YAML file in pod.yaml file

----------------------pod.yaml---------------------

apiVersion: v1

kind: Pod

metadata:

name: my-pod

spec:

containers:

- name: pod-container

image: nginx

ports:

- containerPort: 80

--------------------------------------------------------

1. kubectl apply -f pod.yaml >> to create the pod and will deployed on minikube
2. kubectl describe pod my-pod >> to check the details inside the POD

If you want to add network volume add (pendrive add) >>

1. Write the YAML file

----------------------pod.yaml----------------------

apiVersion: v1

kind: Pod

metadata:

name: my-pod

spec:

containers:

- name: pod-container

image: nginx

ports:

- containerPort: 80

hostPort: 8080

volumeMounts:

- mountPath: /usr/share/nginx/html

name: volume-html

volumes:

- name: volume-sumantra

emptyDir:

medium: "Memory"

sizeLimit: "100Mi"

---------------------------------------------------------------

1. kubectkubectl delete pod my-pod >> Delete running POD first
2. kubectl apply -f pod.yaml >> to create the pod
3. kubectl get pod >> To check the running POD details
4. kubectl describe pod my-pod >> copy the name of POD and paste in this command
5. kubectl exec -it my-pod -- bash >> to get inside the bash shell
6. root@my-pod:/# cd /usr/share/nginx/html >> to go to that path where volume (pendrive) attached
7. create directory and files
8. root@my-pod:/# mkdir sumantra
9. root@my-pod:/# nano file.txt & nano file1.txt & nano file2.txt
10. root@my-pod:/usr/share/nginx/html# kill 1 >> to exit from the bash file
11. Now the docker container has killed means destroyed but here controller will create the container again
12. kubectl get pod >> Now POD has auto created & running