KUBERNETES ARCHITECTURE

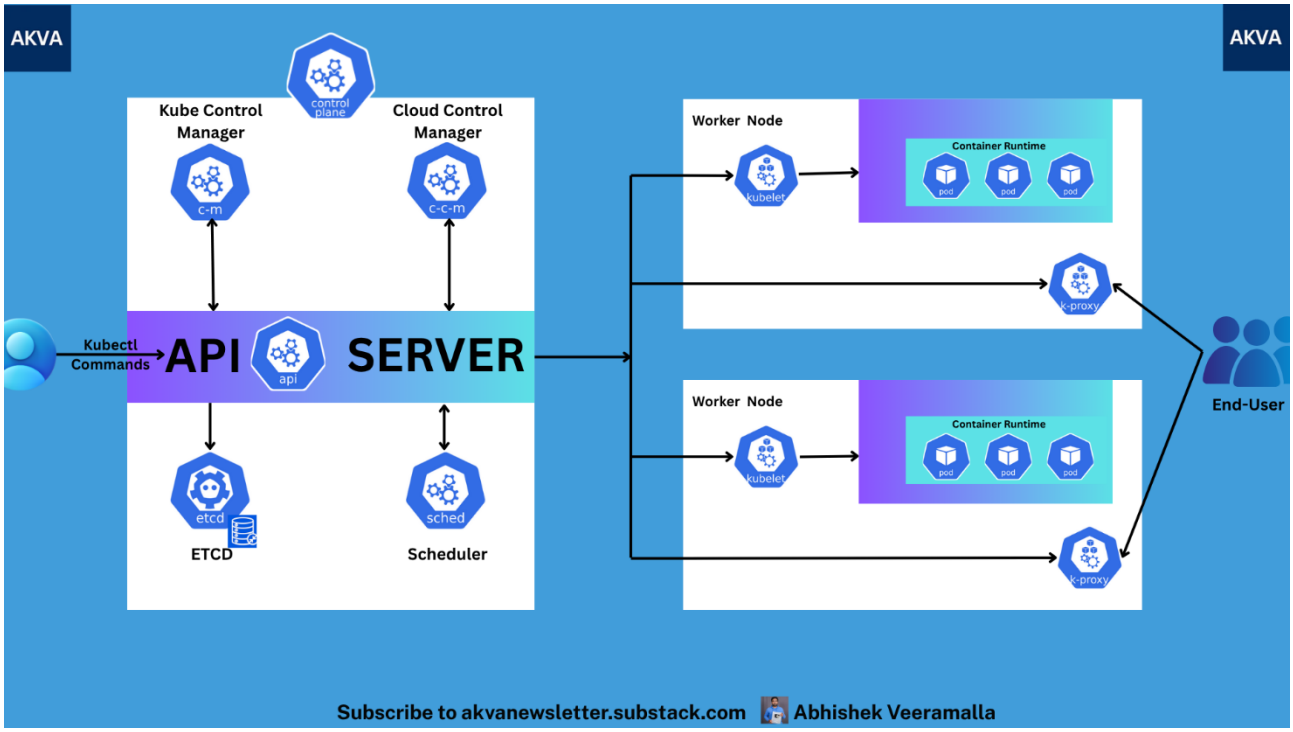
Kubernetes follows a **Master–Worker** node architecture. The smallest deployable unit is a **Pod**, which acts as a lightweight wrapper around one or more containers, adding advanced features like networking and storage.

All requests first go through the **Master Node** (**Control Plane**), which manages and coordinates the cluster, and are then executed by the **Worker Nodes** (**Data Plane**).

Key Roles:

→ Master Node (Control Plane): Manages and controls cluster operations.

→ Worker Node (Data Plane): Runs workloads and executes tasks as directed by the Control Plane.



A pod is deployed over Worker node(Data Plane).

A) Worker Node has following components:

- Kubelet

- Kubeproxy
- Container Runtime

1) Kubelet: The **Kubelet** is responsible for creating and maintaining Pods on a node. It ensures that each Pod remains up and running. If a Pod goes down, the Kubelet leverages Kubernetes' **self-healing** capability by notifying the **API Server** in the Control Plane, which then takes corrective action to restore the Pod.

2) Container Runtime: To run a **Pod**, Kubernetes requires an **execution environment** provided by a **container runtime**. It's not limited to Docker (Dockershim); other runtimes like **containerd** and **CRI-O** are also supported. Kubernetes follows a standard called the **Container Runtime Interface (CRI)** — any runtime that implements this interface can be used within a Kubernetes cluster.

3) Kubeproxy: Kube-Proxy is the Kubernetes component responsible for **networking** within the cluster. It generates and assigns a unique **IP address** to each Pod and manages network routing using **iptables** (or IPVS) on the node. To distribute traffic across Pods, Kube-Proxy provides **built-in load balancing** capabilities by default.

B) Master Node has following components:

- API server
- etcd
- Kube-Scheduler
- Controller Manager
- Cloud Controller manager

1) API Server: The **API Server** is the **core component** of Kubernetes — the heart of the Control Plane. It serves as the **entry point** for all external and internal requests to the cluster. The API Server handles critical tasks such as deciding **which node** a Pod should run on and managing **security and**

configuration instructions. In essence, it exposes the entire Kubernetes cluster to the **outside world** through a unified API interface.

2) Kube-Scheduler: The **Kube-Scheduler** is responsible for **scheduling Pods** and **resources** within a Kubernetes cluster. While the **API Server** determines which node a Pod should run on, the Kube-Scheduler ensures that the resources are properly allocated based on availability and constraints. It receives scheduling information from the API Server and assigns Pods to the most suitable nodes.

3) ETCD: **etcd** is the **backing store** for the entire Kubernetes cluster. It stores all cluster data as **key–value** pairs, representing the **state** and **configuration** of every Kubernetes resource. Without etcd, no cluster information would persist. It plays a crucial role in **cluster recovery**, **backup**, and **upgrades**.

4) Controller Manager: Kubernetes can auto-scale Pods during high traffic using **controllers** like the **ReplicaSet** Controller, which ensures the number of running Pods matches the replica count defined in the YAML file. There are multiple such controllers in Kubernetes, each serving a specific purpose. The **Controller Manager** oversees and manages all these **controllers**, ensuring they function correctly within the cluster.

5) Cloud Controller Manager: When a Kubernetes cluster runs on a **cloud platform** like **AWS** (EKS), **Azure** (AKS), or **GCP** (GKE), the **Cloud Controller Manager (CCM)** acts as a **bridge** between **Kubernetes** and the **cloud provider**. It translates Kubernetes resource requests—such as creating a Load Balancer or persistent storage—into **cloud-specific API calls** that the platform can understand.

If the cluster is deployed **on-premises**, a **CCM** is not required. For a **New Cloud Platform**, its logic must be integrated by submitting a pull request to the open-source CCM repository, enabling Kubernetes to manage resources on that platform.