KUBERNETES INTRODUCTION

Kubernetes (K8s) is an **open-source container orchestration platform**.
Think of it as the “operating system for your data center,” responsible for:

- Deploying containers
- Scaling them
- Healing them if something fails
- Networking them together
- Managing storage
- Rolling out updates gradually

It gives you **infrastructure automation** with **self-healing** and **declarative state management**.

KUBERNETES ARCHITECTURE AT A GLANCE

Kubernetes follows a **master–worker** architecture:

- **Control Plane (Master)** → Think of this as the “brain”
- **Data Plane (Worker Nodes)** → The “muscles” running your apps

Here’s the high-level layout:

```
                  +-------------------------+
                  |      Control Plane      |
                  +-------------------------+
                  | API Server              |
                  | etcd                    |
                  | Scheduler               |
                  | Controller Manager      |
                  | Cloud Controller Mgr    |
                  +-------------------------+
                            |
                 --------------------------------
                 |              |               |
      +-----------------+ +-----------------+ +-----------------+
      |   Worker Node   | |   Worker Node   | |   Worker Node   |
      +-----------------+ +-----------------+ +-----------------+
      | Kubelet         | | Kubelet         | | Kubelet         |
      | Kube-Proxy      | | Kube-Proxy      | | Kube-Proxy      |
      | Container Runtm| | Container Runtm| | Container Runtm|
      +-----------------+ +-----------------+ +-----------------+
      | Actual Pods     | | Actual Pods     | | Actual Pods     |
```

Let’s dive in deeply — and warmly — into each part.

CONTROL PLANE COMPONENTS (The Brain of Kubernetes)

kube-apiserver (The Front Door of Kubernetes)

This is the *heart* of the system.

- Every kubectl CLI call goes **only through this API server**.
- It validates requests.
- Stores data in etcd.
- Exposes a REST API.

Think of it like a receptionist who:

- Checks if you're allowed (authn/authz)
- Validates your request
- Hands the request to the appropriate internal subsystem

It's stateless — you can run multiple replicas behind a load balancer.

---

etcd (The Source of Truth)

etcd is a **distributed key-value store**.

It stores:

- Configurations
- Desired state
- Cluster metadata
- Secrets (encrypted at rest)
- Pod definitions
- Node status

Kubernetes *relies entirely on etcd*.
If etcd is corrupt or lost — your cluster is essentially gone.

That's why large enterprises take **etcd backups** very seriously.

---

kube-scheduler (Decides *where* pods run)

The scheduler watches for **pending pods** and decides which node they should go to.

It evaluates:

- Resource availability (CPU, memory)
- Taints & tolerations
- Node affinity/anti-affinity
- Pod topology constraints
- Pod priority
- Node health
- Existing workloads

It chooses **the best possible node** for each pod.

---

kube-controller-manager

This is a collection of “control loops” that run continuously.

Think of Kubernetes as constantly asking:

> “Is the actual state equal to the desired state?”

If not, controllers take action.

Important controllers:

- **Node controller** → Detects when nodes die
- **Deployment controller** → Ensures correct ReplicaSets
- **ReplicaSet controller** → Manages pod counts
- **Job controller** → Handles batch jobs
- **ServiceAccount controller**
- **Volume controller**

Each controller works like:

```
loop:
   read desired state from etcd
   read actual state from cluster
   if mismatch: fix it
```

---

cloud-controller-manager (Only in cloud environments)

This component integrates Kubernetes with cloud providers like AWS, Azure, GCP.

It manages:

Cloud-specific controllers:

- **Node controller** → Detects cloud VM issues
- **Route controller** → Configures cloud routing
- **Service controller** → Creates ELBs
- **PersistentVolume controller** → Creates storage disks dynamically

Without this, Kubernetes wouldn't be able to request:

- Load balancers
- Cloud disks
- Cloud networking resources

DATA PLANE COMPONENTS (The Muscle of Kubernetes)

Worker nodes are where your containers actually run.

kubelet (Node Agent)

Every node runs a kubelet.

It:

- Talks to the API server
- Ensures the containers *actually* run
- Monitors pod health
- Restarts containers if needed
- Mounts volumes
- Pulls images

If you describe a pod:

```
replicas: 3
```

The kubelet on each node helps ensure that at least 3 pods always exist.

---

kube-proxy (Cluster Networking)

kube-proxy programs the node's network rules so services work.

Depending on mode:

- iptables
- ipvs

Responsibilities:

- Load balance traffic across pod endpoints
- Maintain Service → Pod mappings
- Allow east–west traffic inside the cluster

Even if 50 pods come and go behind a Service, kube-proxy keeps forwarding working reliably.

Container Runtime (Docker, containerd, CRI-O)

Kubernetes doesn't run containers itself — it delegates to a runtime using **CRI (Container Runtime Interface)**.

Supported runtimes:

- containerd (most common)
- CRI-O (popular in OpenShift)
- Docker (deprecated as runtime, but Docker images still OK)

This runtime:

- Pulls images
- Starts containers
- Manages namespaces/Cgroups
- Handles networking via CNI

ADDITIONAL CLUSTER SERVICES

CNI Plugin (Networking layer)

Handles pod networking:

- Assign IP per pod
- Create virtual interfaces
- Handle routes

Popular CNIs:

- Calico
- Weave

- Cilium
- Flannel
- AWS VPC CNI (EKS)

CSI Plugin (Storage layer)

Handles persistent volumes:

- EBS / Azure Disk / GCP PD
- NFS
- Ceph
- Local disks

Helps pods request storage dynamically.

PUTTING EVERYTHING TOGETHER (Flow Example)

Let's say you run:

```
kubectl apply -f nginx-deploy.yaml
```

Here's the flow:

1. **kubectl → API Server**
2. API server writes desired state to **etcd**
3. Scheduler notices:
    - "There is a pod with no node"
4. Scheduler selects nodeA
5. kubelet on nodeA:
    - Pull image
    - Create container
    - Setup networking (via CNI)
6. kube-proxy updates service maps
7. Controller manager watches:
    - Ensures correct number of replicas

You get a healthy deployment with zero manual effort.

VISUAL: COMPLETE ARCHITECTURE

```
                   +-------------------------------+
                   |         Control Plane         |
                   +-------------------------------+
                   |  kube-apiserver               |
                   |  etcd                         |
                   |  kube-scheduler               |
                   |  kube-controller-manager      |
                   |  cloud-controller-manager     |
                   +-------------------------------+
                                 |
                           (REST API)
                                 |
        -------------------------------------------------------
        |                      |                              |
+----------------+     +----------------+              +----------------+
| Worker Node 1  |     | Worker Node 2  |              | Worker Node 3  |
+----------------+     +----------------+              +----------------+
| kubelet        |     | kubelet        |              | kubelet        |
| kube-proxy     |     | kube-proxy     |              | kube-proxy     |
| Runtime        |     | Runtime        |              | Runtime        |
| Pods           |     | Pods           |              | Pods           |
+----------------+     +----------------+              +----------------+
```