Docker Limitations

- **No built-in load balancing:** Docker does not natively support distributing traffic across multiple containers.
- **Lacks self-healing:** If a container crashes, Docker won't automatically restart or replace it without additional tooling.
- **Limited volume management:** Docker provides basic volume capabilities but lacks advanced management features like dynamic provisioning or backups.
- **Simplistic networking:** Docker's networking options are not robust enough for complex or enterprise-level scenarios without external tools.
- **No auto-scaling:** Docker alone cannot dynamically scale containers based on resource usage or traffic demand.
- **Weak secrets and configuration management:** Docker does not offer strong built-in support for managing sensitive data or application configurations securely.

Currently, we build container images with Docker but run them using Kubernetes (K8s). Kubernetes is responsible for running containers, managing network connectivity, handling volume mounts, and ensuring better orchestration and scalability.

Kubernetes

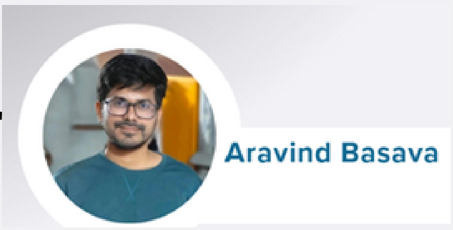
Kubernetes is an open-source container orachestration platform for automating the deployment, scaling and management of containers or pods.

One major advantage of Kubernetes over Docker alone is its ability to run **containerized applications** across multiple virtual machines (VMs), enabling scalability and high availability.

Kubernetes Architecture Overview

Kubernetes consists of:

- **Workstation:** Where you interact with the cluster (e.g., using `kubectl`).
- **Control Plane / Master Node:** Manages the cluster state.
- **Worker Nodes:** Where containers (pods) run.



Control Plane Components

1. **kube-apiserver**

The `kube-apiserver` is the **front-end** of the Kubernetes control plane. It acts as the **main entry point** for all cluster operations.

- Handles **REST API requests** from `kubectl`, internal components (scheduler, controllers), and external tools.
- Validates, authenticates, and authorizes requests.
- Communicates with **etcd**, the cluster's source of truth.
- It is the only component that communicates directly with `etcd`, whereas all other control plane components access `etcd` indirectly through the API server.
- All communication between the user and the Kubernetes cluster **must go through the API ser ver**.

---

2. **etcd**

`etcd` is a **distributed and consistent key-value store** that stores **all cluster data**.

- Holds the entire state of the cluster (nodes, pods, ConfigMaps, Secrets, etc.).
- If `etcd` fails, the control plane cannot function properly.
- Regular backups are critical—typically stored in **cloud storage** like Amazon S3.

---

3. **kube-scheduler**

The `kube-scheduler` assigns newly created pods to suitable nodes by evaluating:

- Available CPU, RAM, and storage
- Taints and tolerations
- Node selectors and affinities

---

4. **kube-controller-manager**

The `kube-controller-manager` ensures the **desired state** matches the **actual state** of the cluster. It runs multiple controllers:

Components:

- **Node Controller** – Detects and responds to node failures.
- **Replication Controller** – Ensures the correct number of pod replicas.
- **Job Controller** – Manages batch jobs and ensures completion.
- **Endpoints Controller** – Maintains service-to-pod endpoint mappings.

5. **cloud-controller-manager**

Integrates Kubernetes with **cloud provider APIs** (e.g., AWS, GCP, Azure).

Responsibilities:

- **Load Balancer Controller** – Provisions cloud load balancers.
- **Persistent Volume Controller** – Manages cloud storage volumes.
- **Node Lifecycle Controller** – Handles cloud-level node deletions or failures.
- **Route Controller** – (Optional) Configures cloud-specific network routes.

🔄 Comparison: `kube-controller-manager` vs `cloud-controller-manager`

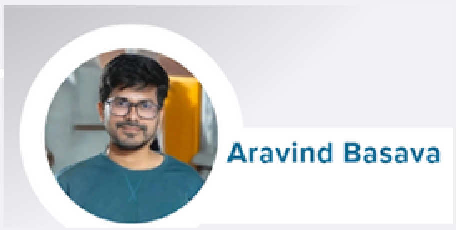
Feature	kube-controller-manager	cloud-controller-manager
Purpose	Manages **cluster-level controllers** (generic to all setups)	Handles **cloud-provider-specific tasks**
Runs Controllers Like	Node, Replication, Job, Endpoints Controllers	Load Balancer, Persistent Volume, Node Lifecycle, Route Controllers
Cloud Dependency	**Cloud-agnostic** (does not depend on cloud provider)	**Cloud-dependent** (integrates with AWS, GCP, Azure, etc.)
Failure Impact	Affects cluster-level object management and state maintenance	Affects cloud resource provisioning and integration
Extensibility	Limited to core Kubernetes objects	Pluggable per cloud provider, makes Kubernetes modular and portable

Node Components

1. `kubelet`

An agent running on each worker node that:

- Ensures pods are running as expected
- Reports node and pod health to the control plane
- Interacts with the container runtime



2. `kube-proxy`

Manages network rules and forwards traffic:

- Enables service-to-pod communication
- Uses `iptables`, `ipvs`, or `eBPF`
- Supports TCP, UDP, and SCTP load balancing

---

3. Container Runtime

The container runtime runs the actual containers. It:

- Pulls container images
- Starts, stops, and manages containers
- Works with `kubelet`

Common runtimes:

- `containerd`
- `CRI-O`
- (Deprecated) `Docker`

Summar y

Component	Role
kube-apiser ver	Frontend for all control plane interactions
etcd	Stores entire cluster state
kube-scheduler	Assigns pods to suitable nodes
kube-controller-manager	Reconciles desired vs actual state
cloud-controller-manager	Manages cloud integration
kubelet	Ensures containers are running on each node
kube-proxy	Manages networking and load-balancing rules
Container Runtime	Runs containers on nodes

Installation and configuration of Kubernetes in workstation

1. eksctl

`eksctl` is a simple CLI tool used to create and manage Kubernetes clusters on Amazon EKS. It automates many cluster setup tasks and simplifies cluster management.

- **Installation:** Follow the instructions at eksctl installation.
- **Sample cluster config (eks.yml):**

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
name: expense
region: us-east-1

managedNodeGroups:
- name: expense
instanceType: m5.large
 desiredCapacity: 3
 spot: true
```

- **Commands:**

```
eksctl  create  cluster  --config-file=<path-to-eks.yml> # Create an EKS cluster
eksctl delete cluster --config-file=<path-to-eks.yml>   # Delete the EKS cluster
```

---

2. kubectl

`kubectl` is the Kubernetes CLI tool used to interact with and manage Kubernetes clusters, pods, services, and other resources.

- **Installation:** Follow instructions at AWS EKS kubectl guide.
- **Common commands:**

```
kubectl api-resources                  # List all available Kubernetes resource types

kubectl get nodes                      # Display all nodes in the cluster

kubectl create namespace <name>       # Create a new namespace

kubectl delete namespace <name>       # Delete an existing namespace

kubectl describe <resource> <name>    # Show detailed info about a specific resource
```