# Kubernetes Introduction for Beginners

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. Understanding its architecture is crucial for effectively managing and utilizing Kubernetes. Here's an overview of the Kubernetes architecture.

Official documentation is here - https://kubernetes.io/docs. Refer this for more information and deepen your understanding in Kubernetes.

## Containers and Docker - Quick Recap

**Containers**

- Definition: Containers are lightweight, portable units that package an application and its dependencies together.
- Isolation: They provide process and environment isolation without the overhead of traditional virtual machines.
- Efficiency: Containers share the host system's kernel and are more resource-efficient, starting up quickly compared to VMs.

**Docker**

- Definition: Docker is a platform that simplifies the creation, deployment, and management of containers.
- Docker Images: These are templates that define the contents of a container, including the application, libraries, and dependencies.
- Docker Engine: The runtime that executes and manages containers.
- Portability: Docker containers can run consistently across various environments, ensuring reliable application performance.

## What is the need of kubernetes?

**While Docker revolutionizes the packaging and deployment of applications, Kubernetes extends this by managing containerized applications at scale.**

**Here’s why Kubernetes is needed:**

### 1. Automatic Scaling

- Challenge: Manually scaling applications to handle variable loads is complex and inefficient.
- Kubernetes Solution: Kubernetes automatically scales the number of running containers based on demand, ensuring optimal resource usage and application performance.

### 2. Service Discovery and Load Balancing

- Challenge: Routing traffic to the appropriate container instances and balancing the load among them can be difficult.
- Kubernetes Solution: Kubernetes provides built-in service discovery and load balancing, directing traffic to healthy containers and distributing it evenly.

### 3. Self-Healing

- Challenge: Monitoring the health of containers and recovering from failures manually is error-prone and time-consuming.
- Kubernetes Solution: Kubernetes continuously monitors the health of containers and automatically restarts or replaces failed containers to maintain application availability.

### 4. Automated Rollouts and Rollbacks

- Challenge: Deploying new versions of applications without downtime and having the ability to revert to previous versions in case of issues.
- Kubernetes Solution: Kubernetes supports rolling updates, gradually deploying changes without affecting the application’s availability, and allows easy rollbacks if problems are detected.

### 5. Resource Management and Optimization

- Challenge: Efficiently utilizing hardware resources while preventing resource contention.
- Kubernetes Solution: Kubernetes schedules containers based on resource requirements and constraints, optimizing resource allocation and utilization across the cluster.

**6. Persistent Storage Management**

- Challenge: Managing stateful applications and ensuring data persistence across container restarts.
- Kubernetes Solution: Kubernetes provides mechanisms to manage persistent storage, allowing containers to mount storage volumes and retain state across restarts and migrations.

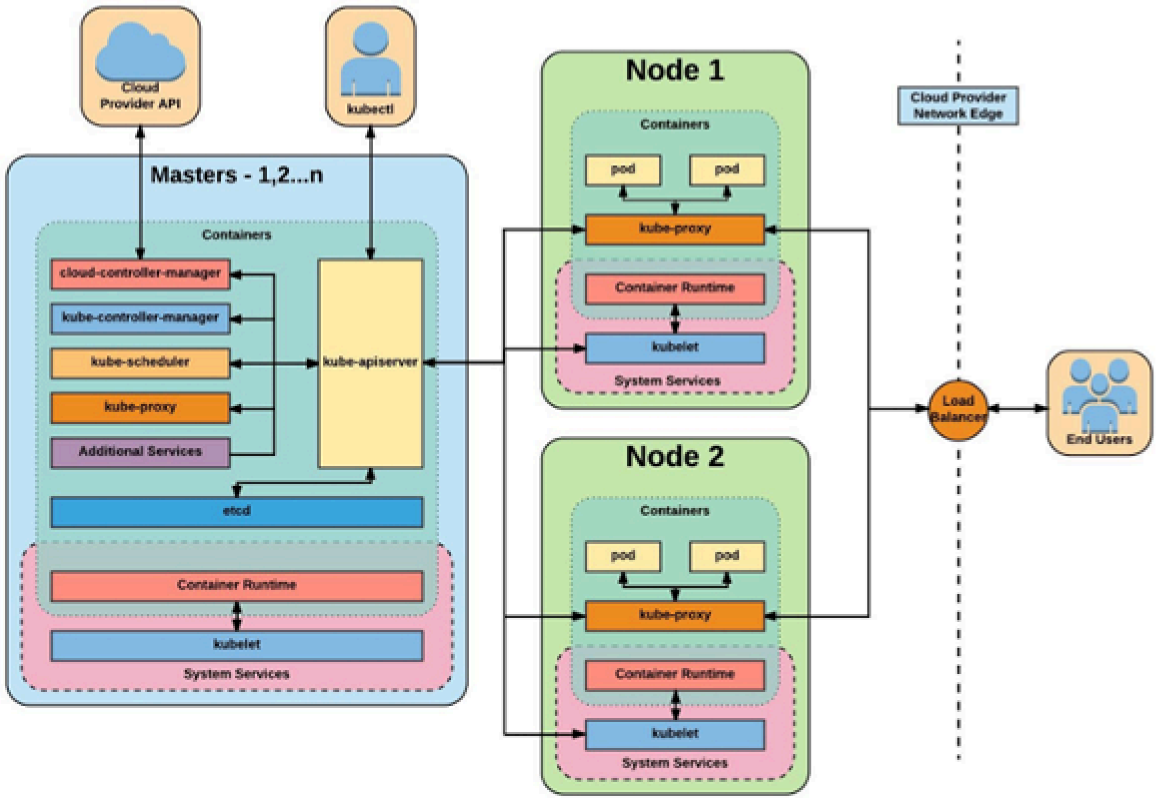
## Advantages of Using Kubernetes with Docker

- Enhanced Scalability: Automatically scale applications horizontally to handle varying loads.
- High Availability: Ensure application resilience through self-healing and replication.
- Operational Efficiency: Automate deployment processes, reducing manual intervention and errors.
- Consistent Environment: Maintain consistent environments across development, testing, and production.
- Resource Optimization: Efficiently allocate resources to maximize hardware utilization and minimize costs.

## Disadvantages of Kubernetes

- Complexity: Kubernetes introduces additional complexity and a steeper learning curve compared to using Docker alone.
- Overhead: Requires additional resources to run Kubernetes control plane components.
- Initial Setup: Setting up and configuring a Kubernetes cluster can be time-consuming and challenging for beginners.

## Look at a typical Kubernetes architecture and its various components.



## 1. Kubernetes Cluster Components

A Kubernetes cluster consists of two main types of components: the control plane and the worker nodes.

### Control Plane

The control plane manages the Kubernetes cluster. Its components include:

- API Server (kube-apiserver): This is the front-end for the Kubernetes control plane. It exposes the Kubernetes API and is the primary entry point for all administrative tasks.
- etcd: This is a key-value store used to store all cluster data. It holds the state of the cluster and ensures that data is reliably stored.

- Scheduler (kube-scheduler): This component assigns workloads (Pods) to worker nodes. It considers resource availability and other constraints to make scheduling decisions.
- Controller Manager (kube-controller-manager): This runs various controllers that handle routine tasks in the cluster, such as:
  - Node Controller: Monitors node status.
  - Replication Controller: Ensures that the specified number of pod replicas are running.
  - Endpoint Controller: Manages endpoint objects, joining services and pods.
  - Service Account & Token Controllers: Manage service accounts and tokens for accessing the API.

#### Worker Nodes

Worker nodes run the containerized applications. Each node includes the following components:

- Kubelet: An agent that runs on each node in the cluster. It ensures containers are running in a Pod and communicates with the control plane.
- Kube-proxy: A network proxy that maintains network rules on nodes. It allows for communication between pods and services both inside and outside the cluster.
- Container Runtime: The software responsible for running containers. Examples include Docker, containerd, and CRI-O.

## 2. Kubernetes Objects

Kubernetes uses a set of abstractions (objects) to represent the state of the cluster:

- Pod: The smallest and simplest Kubernetes object. A pod represents a single instance of a running process in the cluster, which can contain one or more containers.
- Service: An abstraction that defines a logical set of pods and a policy by which to access them. Services enable communication between different parts of an application without being aware of pod IP addresses.
- Volume: A directory containing data, accessible to the containers in a pod. Kubernetes supports various volume types, such as persistent volumes, hostPath, and emptyDir.

- Namespace: A way to divide cluster resources between multiple users (via resource quotas).
- ConfigMap and Secret: These objects are used to inject configuration data and sensitive information into containers.

## 3. Deployment and Scaling

Kubernetes provides controllers to manage application deployment and scaling:

- Deployment: A controller that provides declarative updates to applications. It manages the rollout and rollback of applications, ensuring the desired number of replicas are running.
- StatefulSet: Used for stateful applications that require persistent storage and unique network identifiers.
- DaemonSet: Ensures that a copy of a pod is running on all (or some) nodes in the cluster.
- Job and CronJob: Job is a controller that runs a pod to completion, while CronJob schedules jobs to run at specified times.

## 4. Networking

Kubernetes networking allows for communication between nodes and pods:

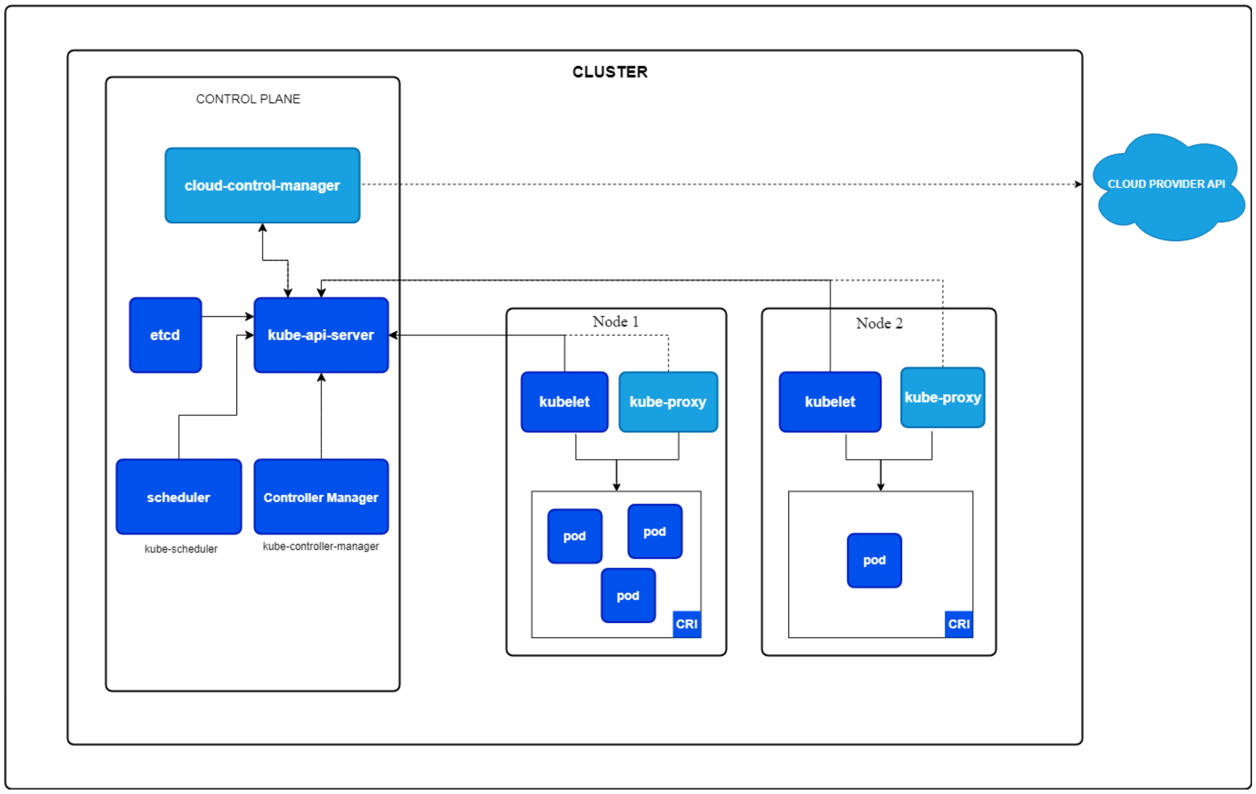
- Cluster Networking: Every pod gets its own IP address, and pods can communicate without NAT.
- Service Networking: Kubernetes services enable load balancing across a set of pods.

## 5. Security

Security in Kubernetes is managed via several features:

- RBAC (Role-Based Access Control): Controls who can access the Kubernetes API and what actions they can perform.
- Network Policies: Control the communication between pods.
- Secrets: Store and manage sensitive information, such as passwords and keys.

## 6. Architecture



## Source: https://kubernetes.io/docs/concepts/architecture/

## Conclusion

Kubernetes architecture is designed to ensure high availability, scalability, and efficient resource management. By understanding its components and how they interact, you can effectively deploy and manage applications in a Kubernetes cluster.