# INTRODUCTION TO **KUBERNETS**

CONTAINER ORCHESTRATION TOOL

**PRESENTED BY**
VIGNESH S

# Kubernetes

*“Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform.”*

**It's designed to automate the deployment, scaling, and management of containerized applications.**

# Kubernetes

It provides a container runtime, container orchestration, container-centric, infrastructure orchestration, self-healing mechanisms, service discovery, load balancing and container (de)scaling.

Initially developed by Google, for managing containerized applications in a clustered
environment but later donated to CNCF

# Certified Dictributions

- Cloud Managed: EKS by AWS, AKS by Microsoft, and GKE by Google

- Self-Managed: OpenShift by Redhat and Docker Enterprise

- Local dev/test: Micro K8s by Canonical, Minikube

# Significance

*"Kubernetes has gained immense popularity because it simplifies complex container management tasks."*

*It's used by tech giants and startups alike to build, deploy, and scale applications efficiently.*

# Importance of Containerization

Before we dive deeper into Kubernetes, let's understand the concept of containerization.

*"Containerization is the practice of packaging an application and its dependencies together into a single unit called a container.”*

# Why Containerization Matters

*Containers provide consistency and portability across different environments, from development to production.*

*They isolate applications, ensuring they run reliably, regardless of the underlying infrastructure.*

# Container Orchestration

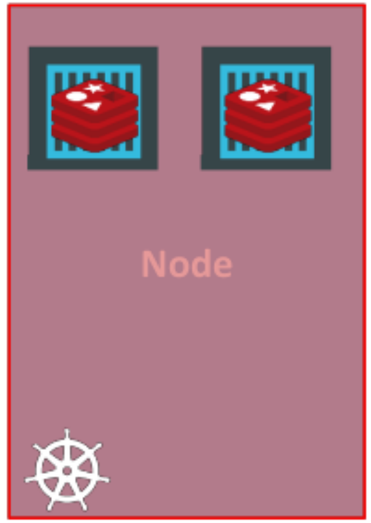
*Container orchestration is the process of automating the deployment, scaling, and management of containerized applications.*

In today's world, applications are complex and need to run on multiple containers across various servers.

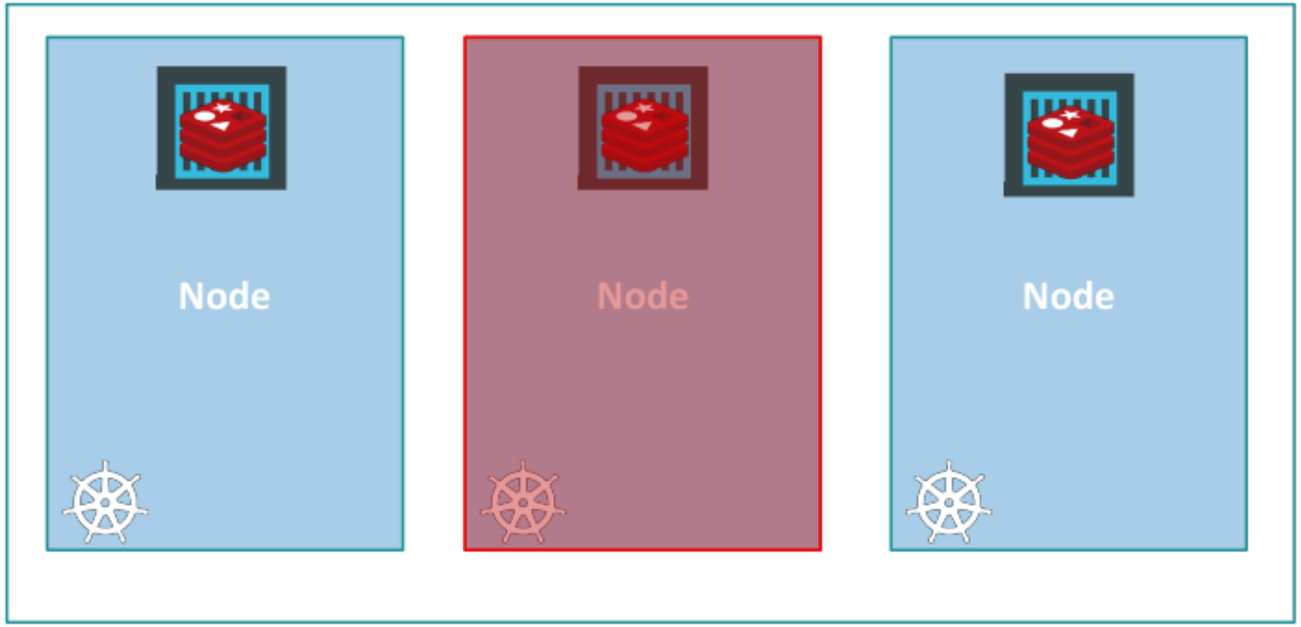
# WHY K8’s

- *"Kubernetes offers automatic scaling, ensuring your applications run efficiently."*
- *"It provides self-healing capabilities, automatically replacing failed containers."*
- *"K8s supports rolling updates for seamless application upgrades."*
- *"It offers load balancing for distributing traffic across containers."*
- *"Kubernetes simplifies storage management with its storage orchestration."*
- *"You can define your application's desired state, and Kubernetes will maintain it."*

# Nodes(Minions)



# Cluster



# Kubernetes Cluster

A Kubernetes cluster is a set of physical or virtual machines and other infrastructure resources that are needed to run your containerized applications. Each machine in a Kubernetes cluster is called a node.

There are two types of nodes in each Kubernetes cluster:

Master node(s): hosts the Kubernetes control plane components and manages the cluster

Worker node(s): runs your containerized applications

# Kubernetes Components

*Kubernetes has two main components: the control plane and nodes.*

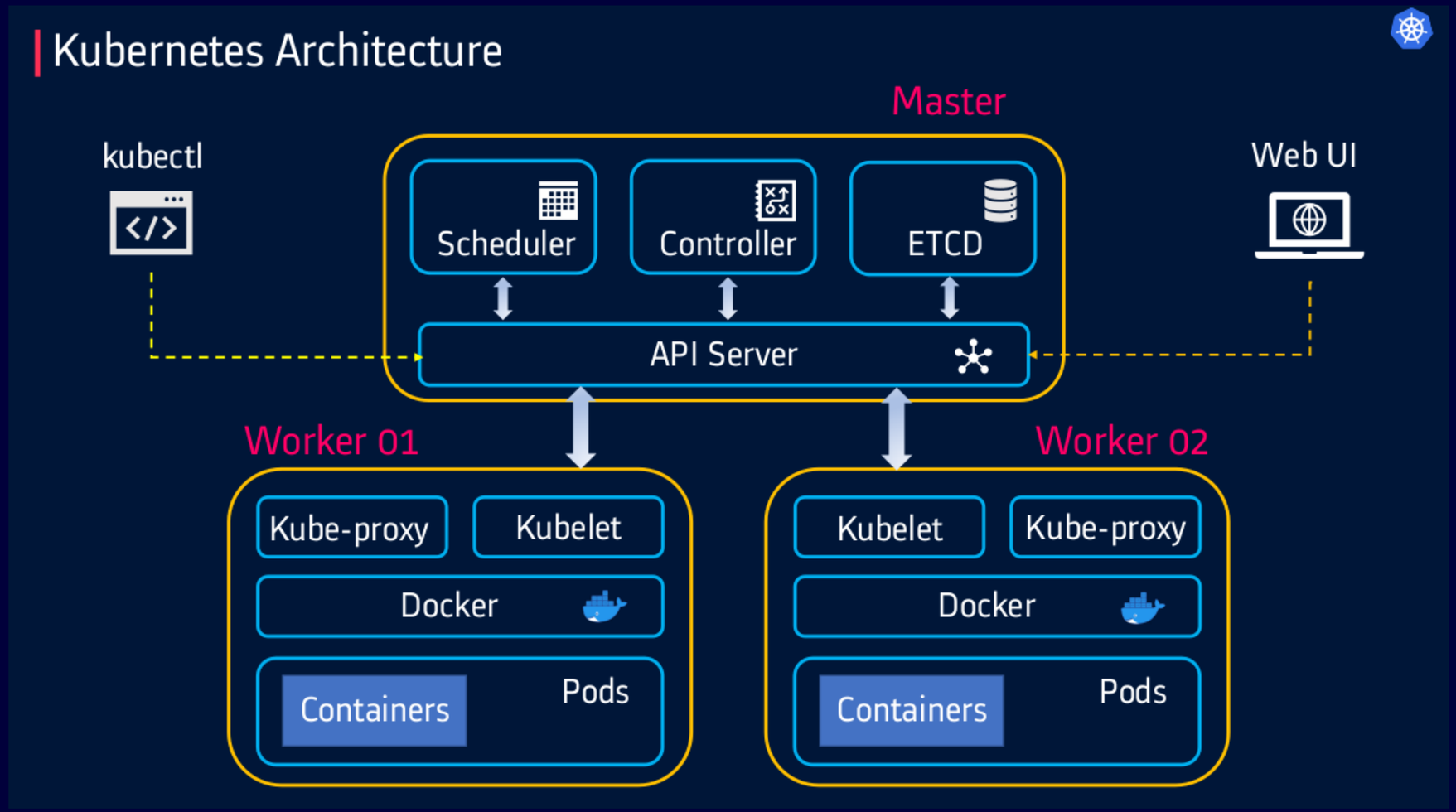
- *Control Plane:*

*The control plane manages the overall cluster state and makes global decisions.*

- *Nodes:*

*Nodes are the worker machines where containers run.*
*Nodes communicate with the control plane to maintain the desired state.*

# Kubernetes Architecture



# KUBERNETS MASTER

Master is responsible for managing the complete cluster.

- You can access master node via the CLI, GUI, or API

- The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes

- For achieving fault tolerance, there can be more than one master node in the cluster.

# KUBERNETS MASTER

It is the access point from which administrators and other users interact with the cluster to manage the scheduling and deployment of containers.

It has four components:

- ETCD
- Scheduler
- Controller
- API Server

# ETCD

• ETCD is a distributed reliable key-value store used by Kubernetes to store all data used to manage the cluster.

• It's designed to be highly available and consistent, which means it ensures that all the information it stores is up to date and accessible even if some parts of the system fail.

• Think of etcd as the "memory" of your Kubernetes cluster. It stores all the important information about what should be running, where it should be running, and how it should be configured.

# Scheduler

The Kubernetes Scheduler is a component responsible for deciding which node (machine) in a Kubernetes cluster should run a newly created pod (a unit of deployment).

Technically, the scheduler listens for new pod specifications and, based on the available information, chooses the most suitable node for each pod. It’s like a manager, making sure that the right pods are assigned to the right nodes to keep everything running smoothly in the cluster

The scheduler doesn't interact directly with containers or pods; instead, it communicates with the Kubernetes API server to assign pods to nodes.

# API server (KUBE-API)

The Kubernetes API Server is a core component of the Kubernetes control plane. It acts as the front-end for the Kubernetes control plane and provides a RESTful API  used to manage the entire Kubernetes cluster.

When you interact with your Kubernetes cluster you're actually communicating with the Kubernetes API server. The API server validates and processes your requests, ensuring that your cluster's state is consistent and secure.

It's responsible for actions like creating or updating pods, services, deployments, and more. Think of the Kubernetes API server as the receptionist at the entrance of a high-tech building, and the building represents your Kubernetes cluster.

# Controller manager

The controllers are the brains behind orchestration.

It runs various controller processes that regulate the state of the system to achieve the desired configuration.

For example, there are controllers for managing replica sets, nodes, endpoints, and more. These controllers constantly watch the cluster's current state and take actions to ensure it matches the desired state as defined by the user.

These controllers work tirelessly in the background, making sure that if something goes wrong or a change is needed (like scaling an application), the cluster automatically corrects itself to stay in the desired state.

# Kubectl

kubectl is a command-line tool used to interact with Kubernetes clusters.

kubectl allows users to send commands to the Kubernetes API server, Users can use kubectl to create, modify, or delete Kubernetes resources like pods, services, deployments, and more.

It also provides features for inspecting cluster resources, managing configurations, and troubleshooting.

- kubectl run nginx used to deploy an application on the cluster.
- kubectl cluster-info is used to view information about the cluster and the
- kubectl get nodes used to list all the nodes part of the cluster.

# Kubernetes Worker

A Kubernetes worker node, often referred to as a "node" or "minion," is a part of a Kubernetes cluster responsible for running containers and managing the workload. Each worker node hosts one or more containers within pods.

COMPONENTS

- Container Runtime
- Kubelet
- Kube Proxy

# Container runtime

Software responsible for running and managing containers.

Containers are lightweight, isolated environments that package applications and their dependencies, allowing them to run consistently across different computing environments.

The container runtime interacts with the host operating system's kernel to create, start, stop, and manage containers.

The most Popular Container Runtime is DOCKER and ContainerD

# KUBELET

A Kubelet is a vital component of a Kubernetes worker node (also known as a minion).

It's an agent that runs on each worker node and communicates with the Kubernetes control plane (master node) to ensure that containers (in the form of pods) are running and healthy on that node.

Think of a Kubelet as a caretaker or supervisor for each worker node in a Kubernetes cluster taking care of tasks like starting, stopping, and monitoring containers as instructed by the control plane.

# Kube Proxy

Kube Proxy is a network proxy component in Kubernetes. It runs on each worker node and is responsible for managing network communication to and from pods

It helps with routing traffic between different pods and services within the cluster. It also enables network address translation (NAT) for pods to communicate with external networks.

The kube-proxy is responsible for ensuring network traffic is routed properly to internal and external services as required and is based on the rules defined by network policies in kube-controller-manager and other custom controllers.

# PODS

Basic scheduling unit in Kubernetes. Pods are often ephemeral

Kubernetes doesn’t run containers directly; instead, it wraps one or more containers into a higher-level structure called a pod

It is also the smallest deployable unit that can be created, scheduled, and managed on a Kubernetes cluster. Each pod is assigned a unique IP address within the cluster.

Pods can hold multiple containers as well, but you should limit yourself when possible. Because pods are scaled up and down as a unit, all containers in a pod must scale together, regardless of their ndividual needs. This leads to wasted resources

# Imperative vs Declarative commands

Kubernetes API defines a lot of objects/resources, such as namespaces, pods, deployments,
services, secrets, config maps etc.

IMPERATIVELY

- Involves using any of the verb-based commands like kubectl run, kubectl create, kubectl
expose, kubectl delete, kubectl scale, and kubectl edit
- Suitable for testing and interactive experimentation

DECLARATIVELY

- Objects are written in YAML files and deployed using kubectl create or kubectl apply
- Best suited for production environments

# Manifest / Spec file

K8s object configuration files - Written in YAML or JSON

EXAMPLE

apiVersion - version of the Kubernetes API
used to create the object
kind - kind of object being created
metadata - Data that helps uniquely identify
the object, including a name and
optional namespace
spec - a configuration that defines the desired for
the object

# Pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: Demo-pod
  labels:
    app: my-app
spec:
  containers:
  - name: nginx-container
    image: nginx:latest
    ports:
    - containerPort: 80
```