kubernetes

Orchestration tool offers

- High Availability or zero down timing – self healing
- Scalability
- Disaster recovery –backup and restore

Docker vs Kubernates

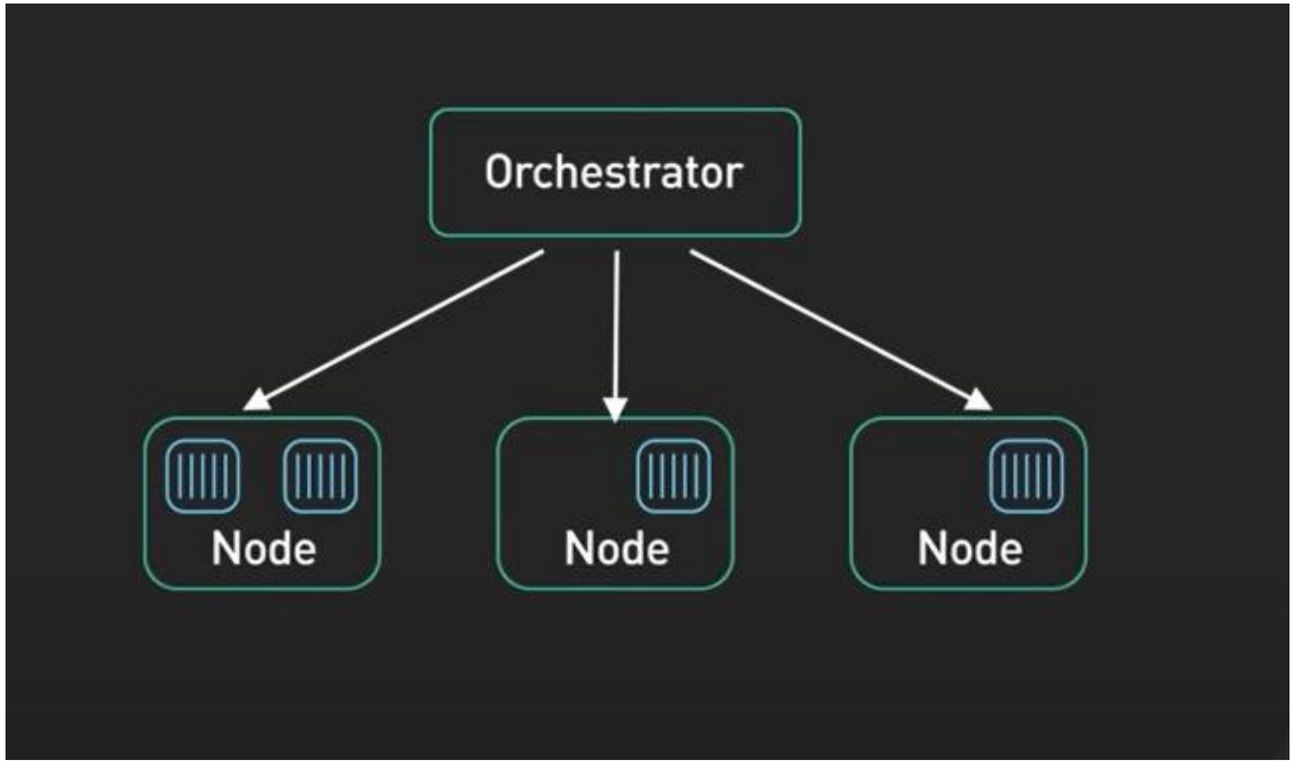
- Docker = container engine → makes it easy to package an app.
- Kubernetes = container orchestrator → makes it easy to run many Docker containers reliably in production.
- Docker = shipping containers (pack your app in a container).
- Kubernetes = port manager (coordinates hundreds of containers across multiple ships/nodes, directs traffic, replaces lost containers).

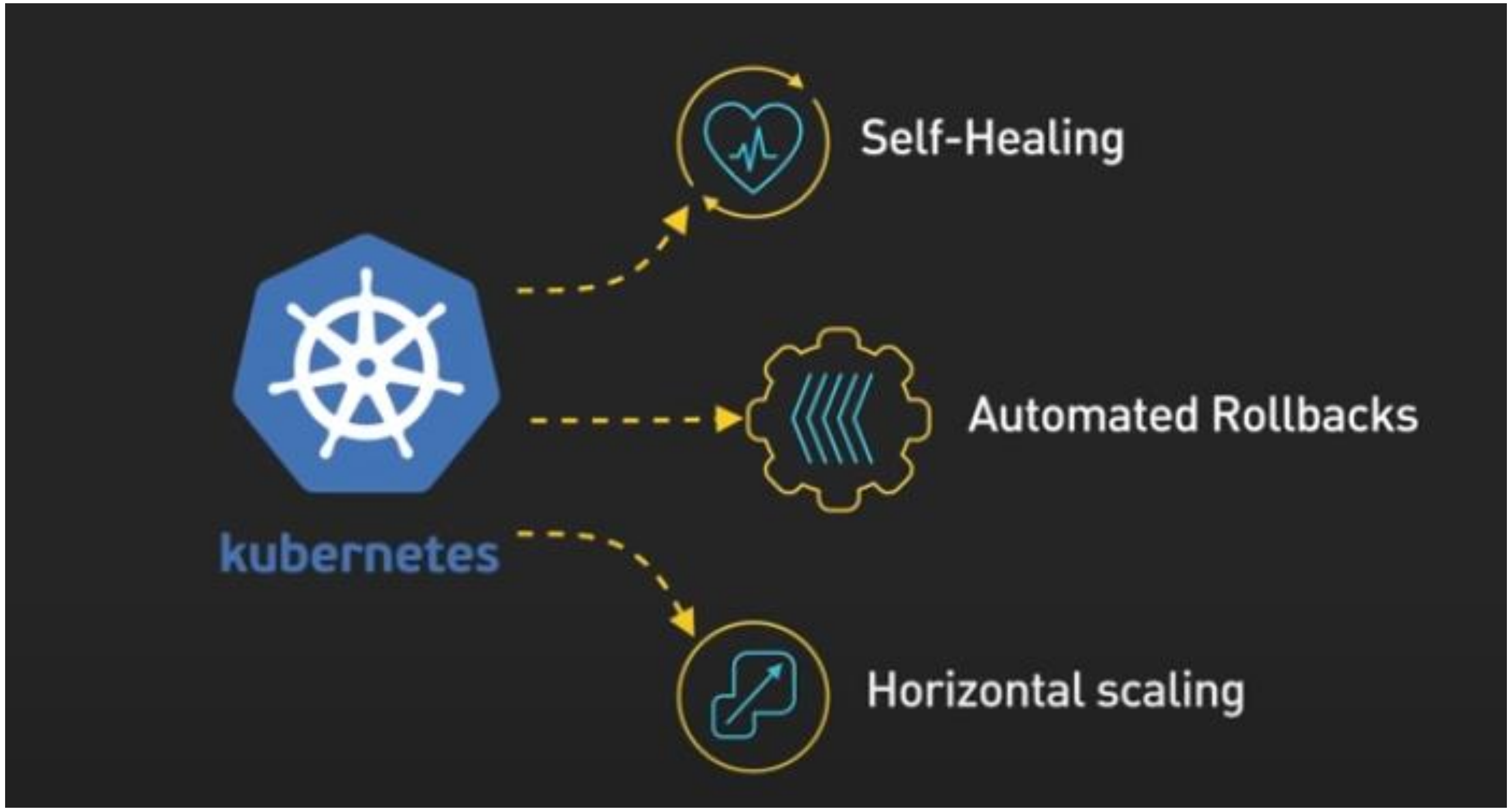
Docker vs Kubernates

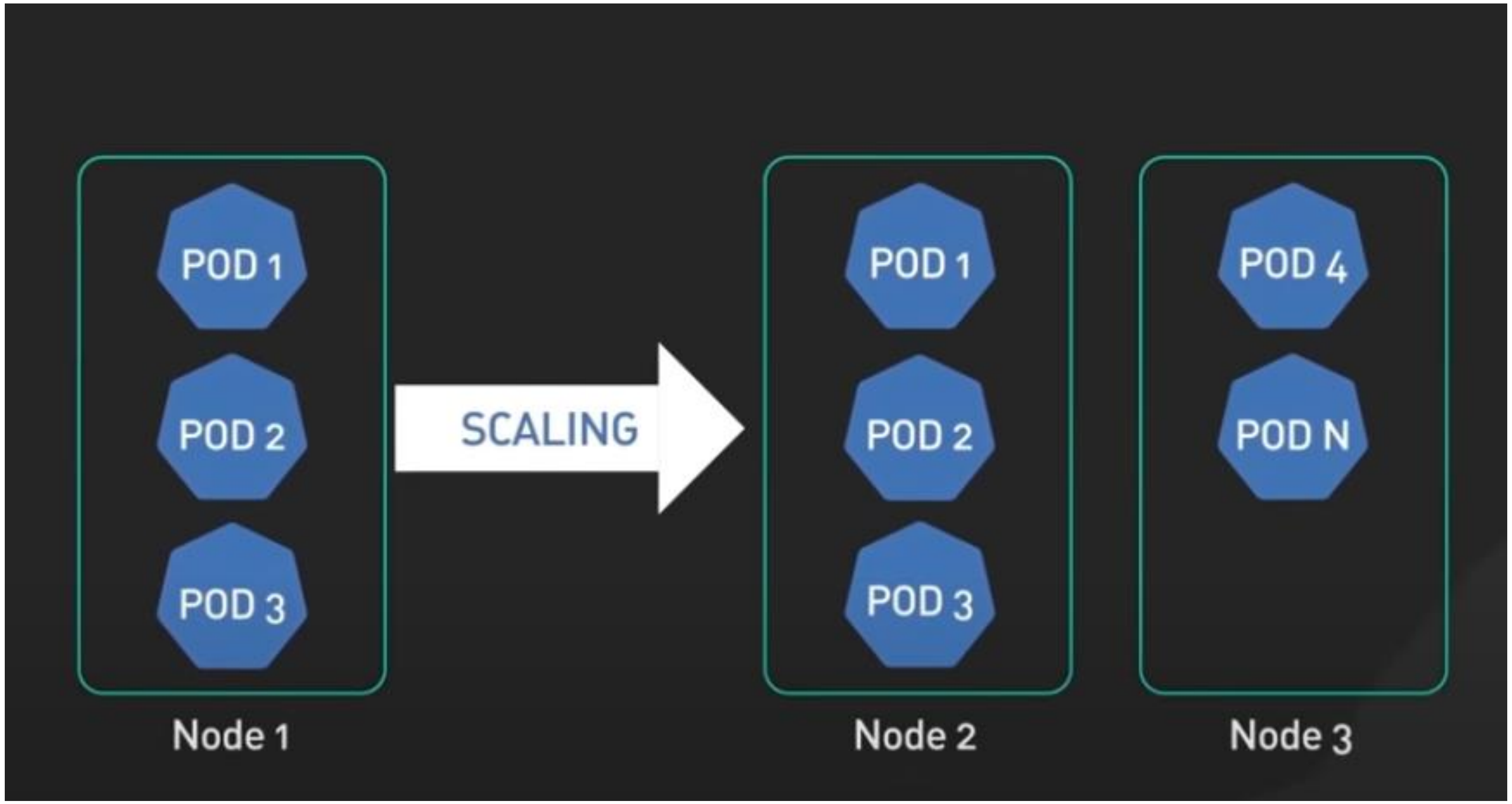
- Kubernetes can use Docker as the container runtime, but it can also use containerd, CRI-O, etc.

- So Docker is a tool, Kubernetes is the system that manages many such tools together.

- Kubernates is the open source container orchestration platform




kubernetes
Self-Healing
Automated Rollbacks
Horizontal scaling

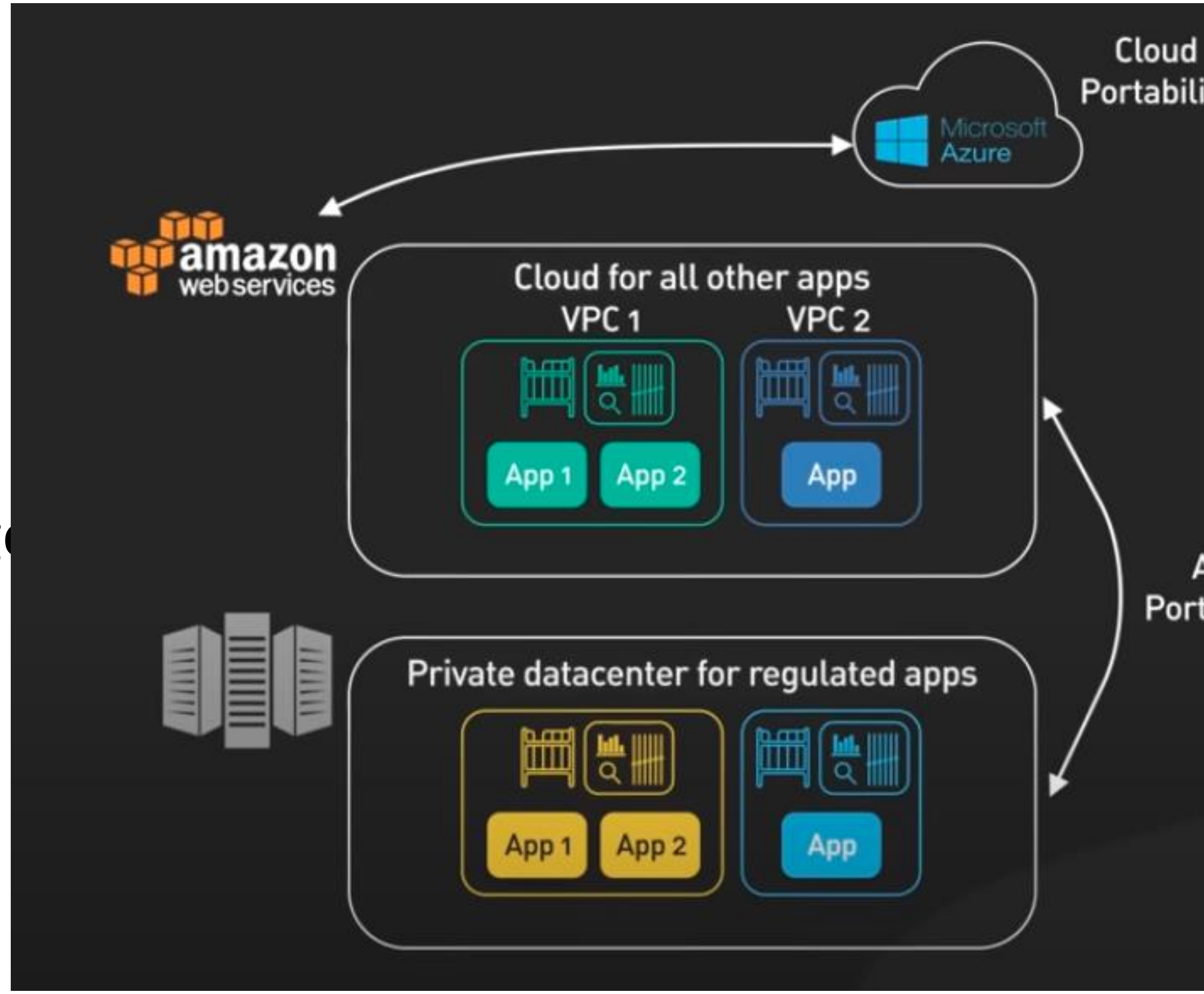


- It is easy to scale the application when needed, allowing us to respond to changes in demand quickly

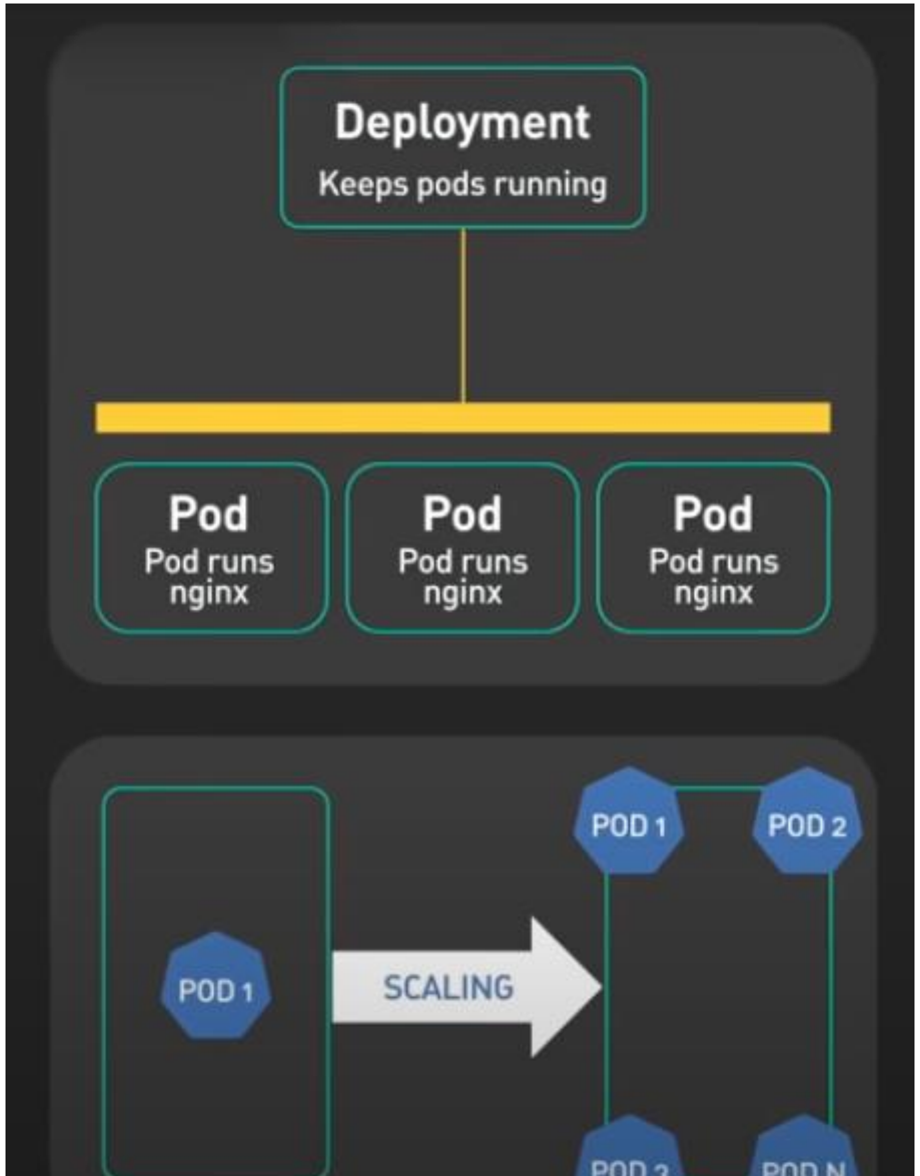
- It helps to deploy applications in consistent and reliable way regardless underlying infrastructure

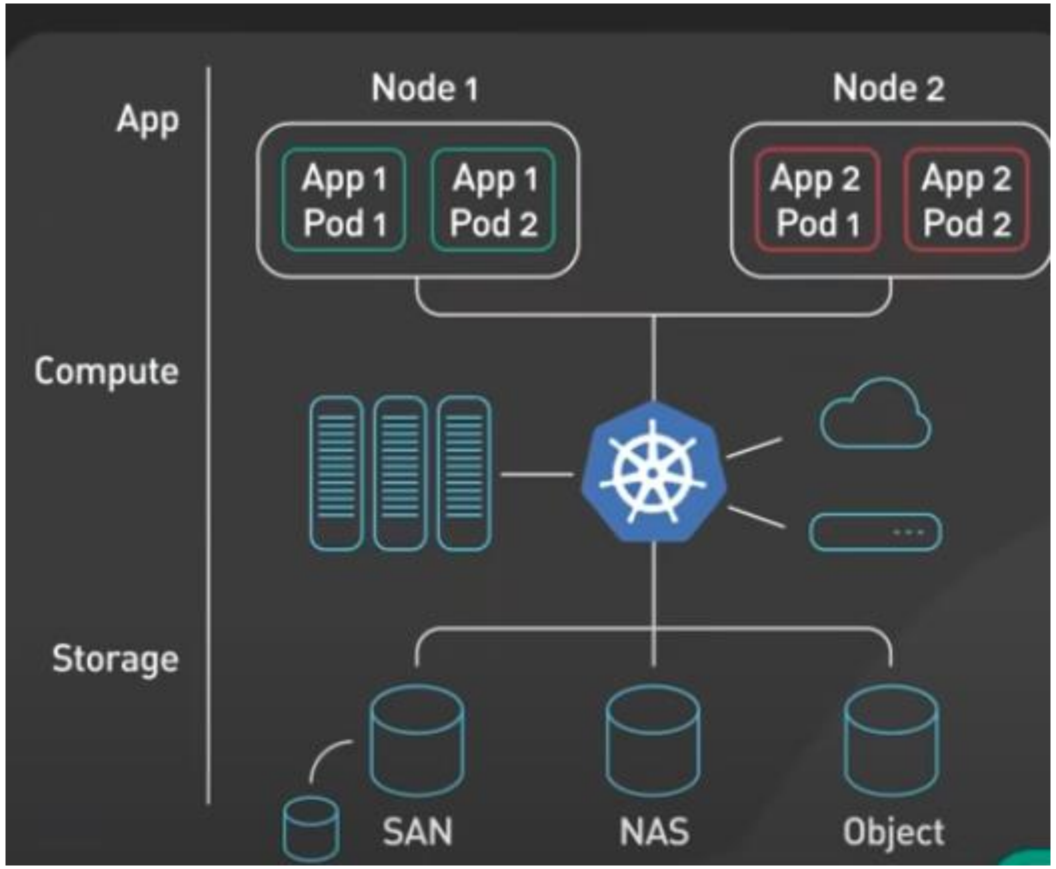
It provides uniform way to packag

deploy applications



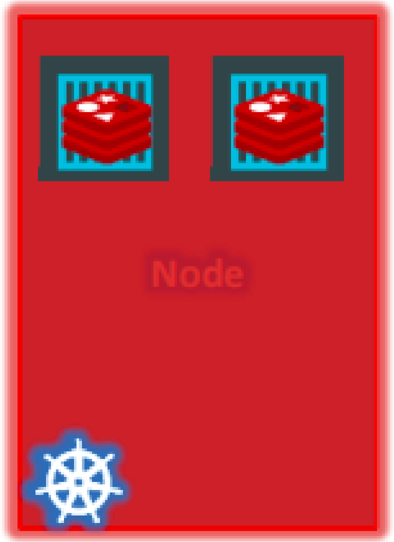
It automates deployments, scaling and management containerized applications





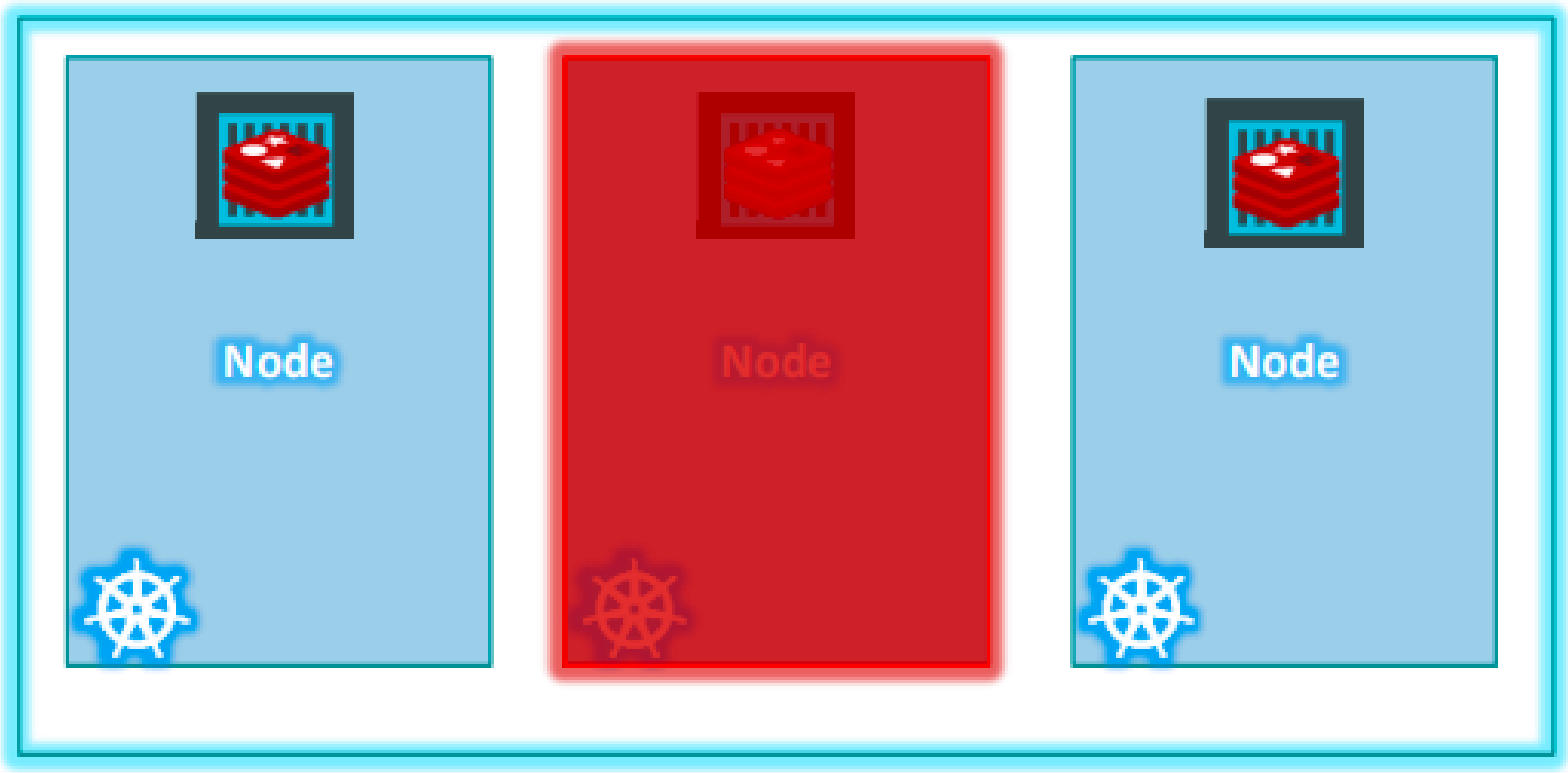
Node

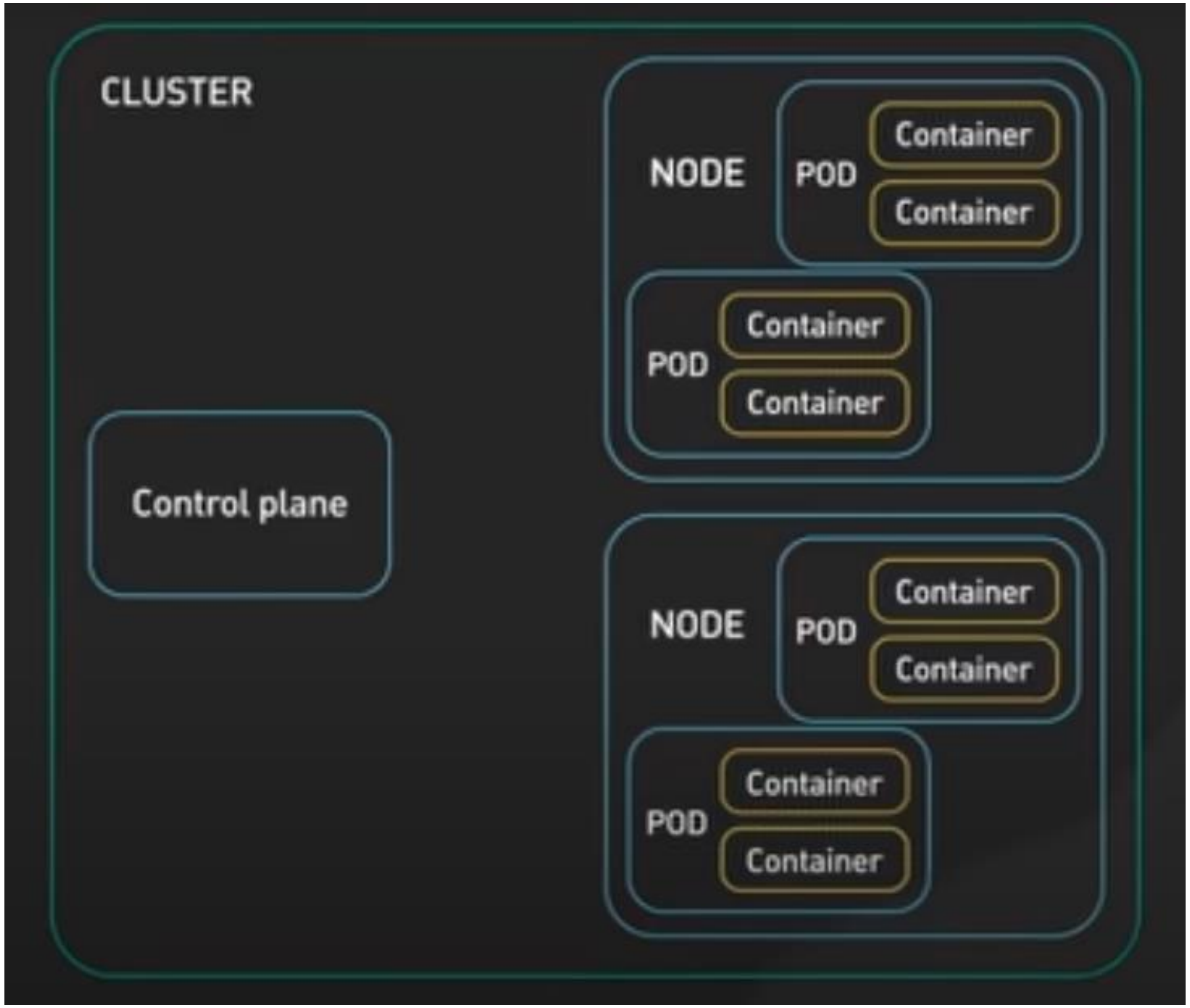
- Let us start with Nodes. A node is a machine – physical or virtual – on which kubernetes is installed. A node is a worker machine and this is were containers will be launched by kubernetes



Cluster

- A cluster is a set of nodes grouped together. This way even if one node fails you have your application still accessible from the other nodes. Moreover having multiple nodes helps in sharing load as well.





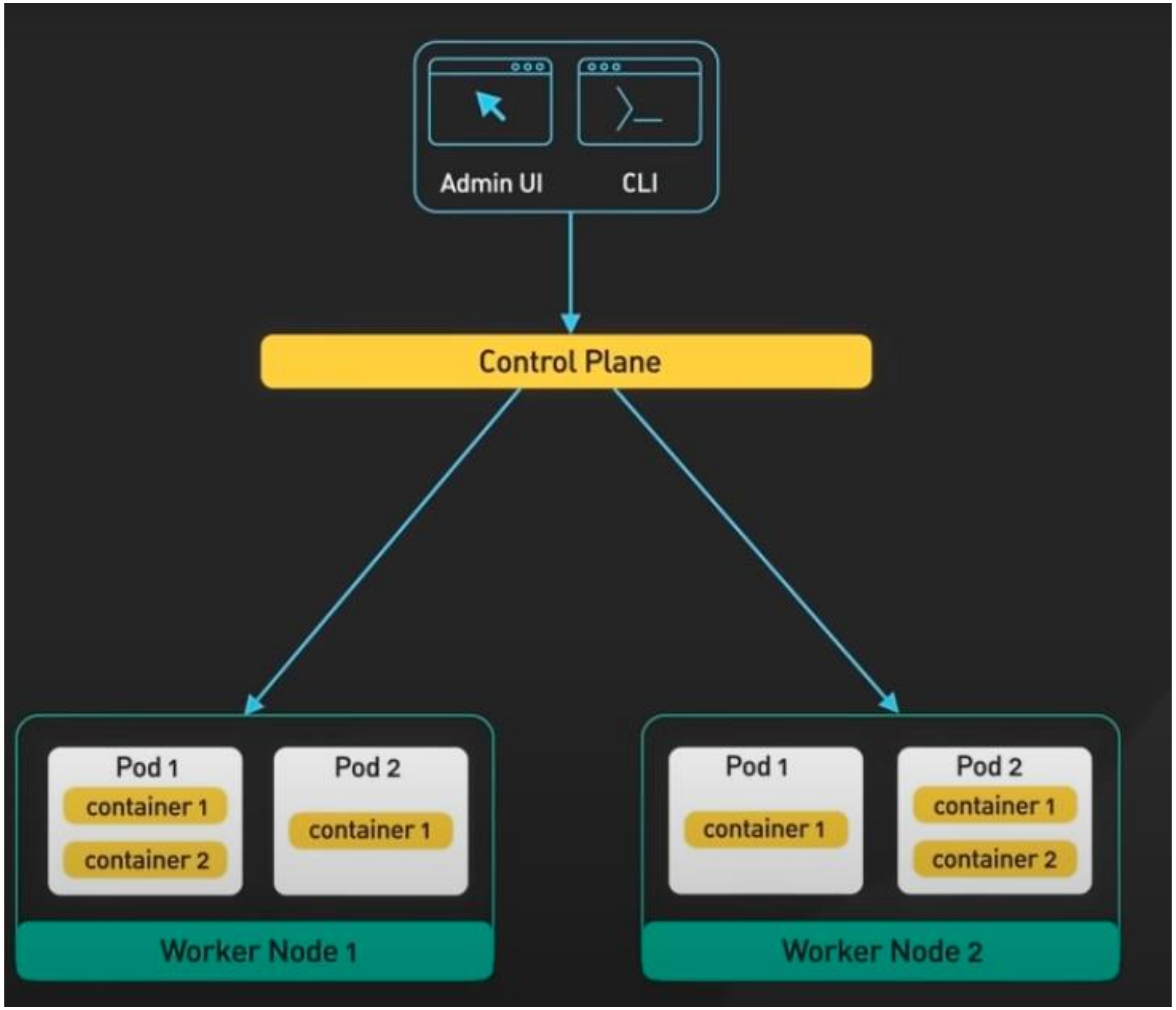
Kubernates cluster has set of machines called node, it is to set run containerized applications

- Kubernetes is divided into **two main parts**:
- 1. Control Plane / Master node(the “brain”)

  -manages the entire cluster

  -decides what to run, where and how
- 2. Worker Nodes (the “muscles”)

  -These are the machines (VMs or physical servers) that actually run the application workloads (Pods).

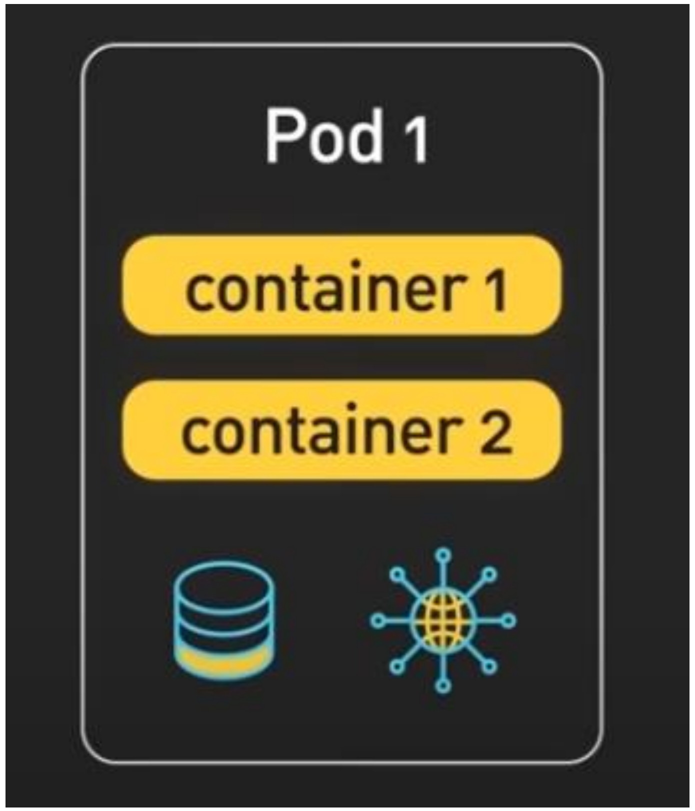

Admin UI
CLI
Control Plane
Pod 1
container 1
container 2
Pod 2
container 1
Worker Node 1
Pod 1
container 1
Pod 2
container 1
container 2
Worker Node 2

Pod

- **Pod** → containerized application run in a pod
 a smallest unit (one or more containers).
Smaller deployable units

Pod host one or more containers
They provide shared storage and networking

Pod is created and managed by control Pane
Pod is basic building blocks of K8s

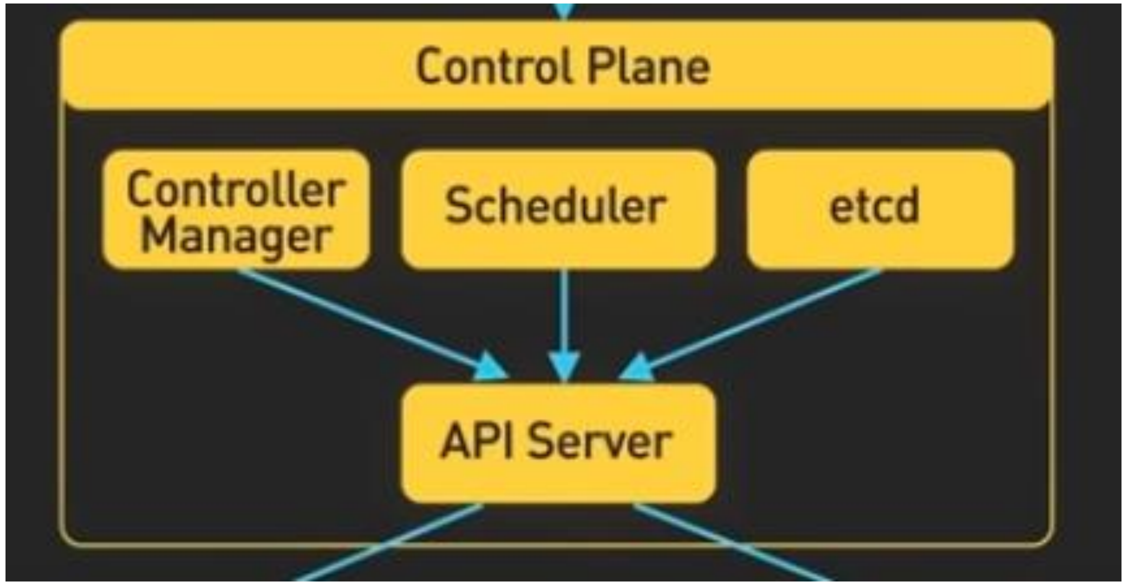


Control Plane

- Api server is the primary

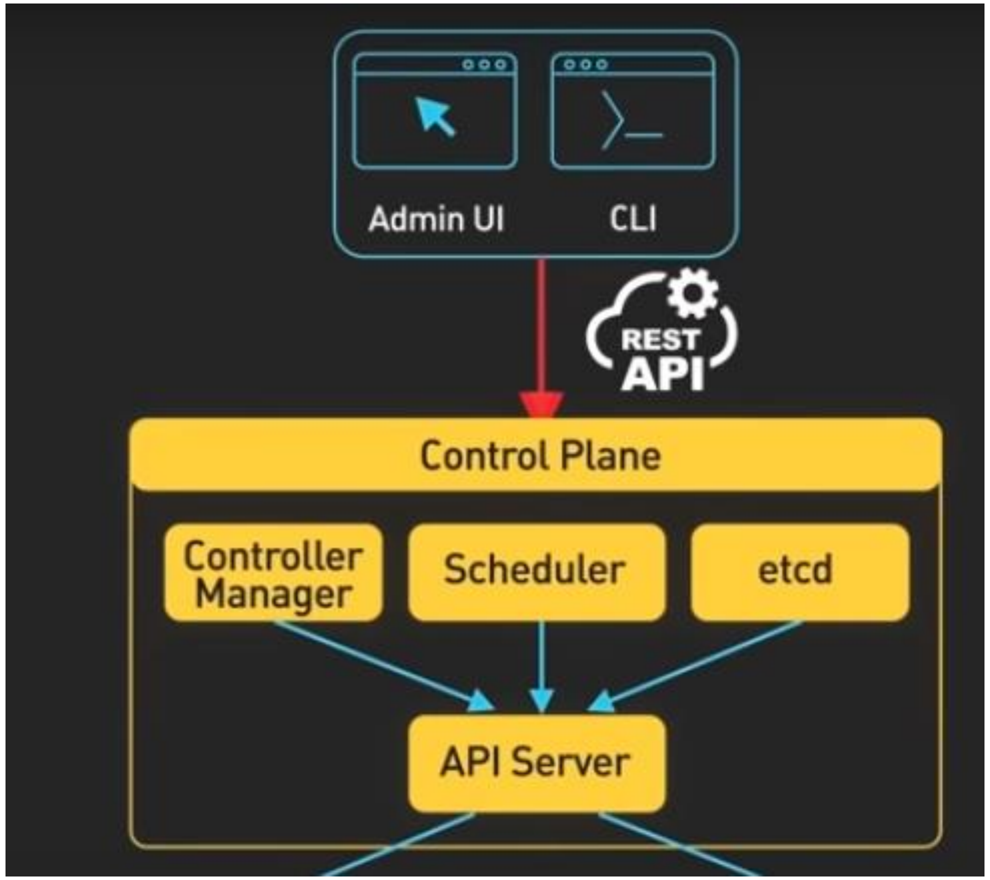
Interface between control

Plane and rest of the clusters



- Control plane exposes a rest api

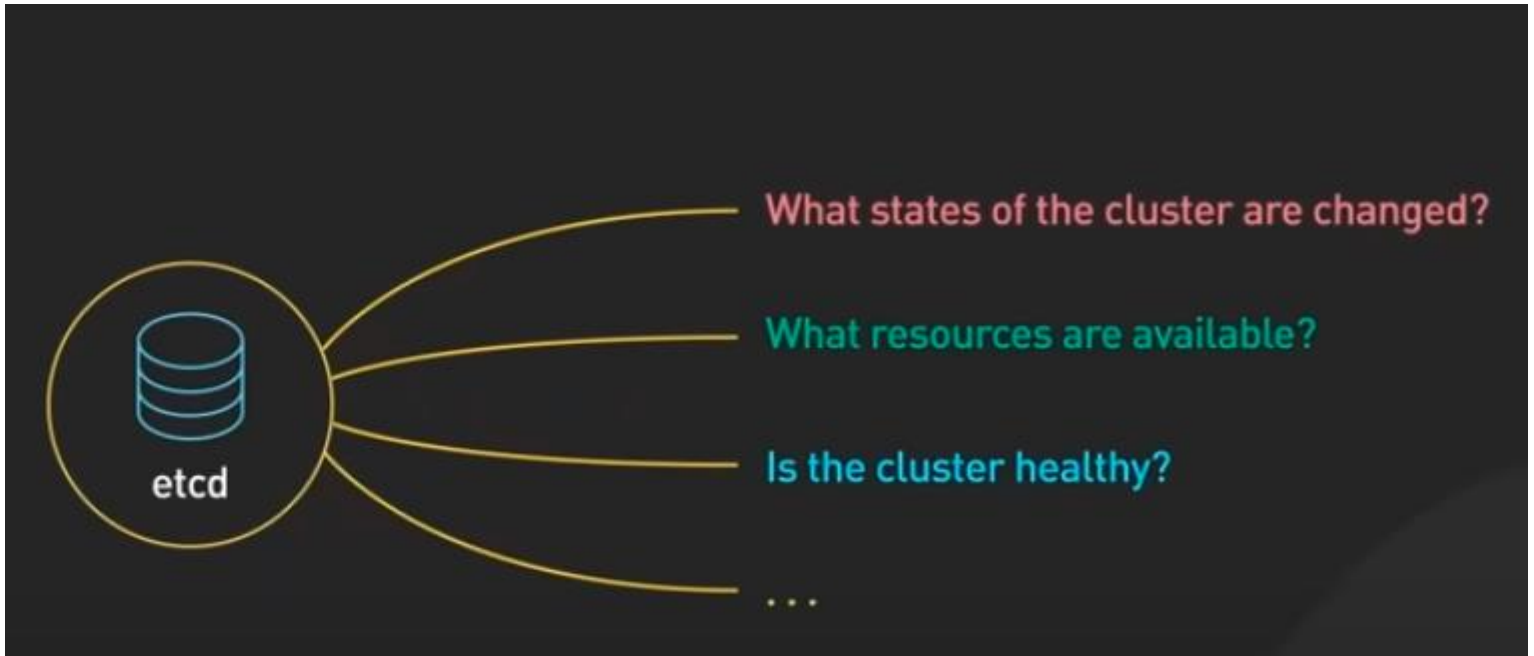
to allow client to interact with control
 plane



etcd

- A distributed key-value store
- It stores clusters persistent state

•Information about **Nodes** (which ones exist, their health).
•**Pods** (what is running, where it is scheduled).
•And other components can store and retrieve information

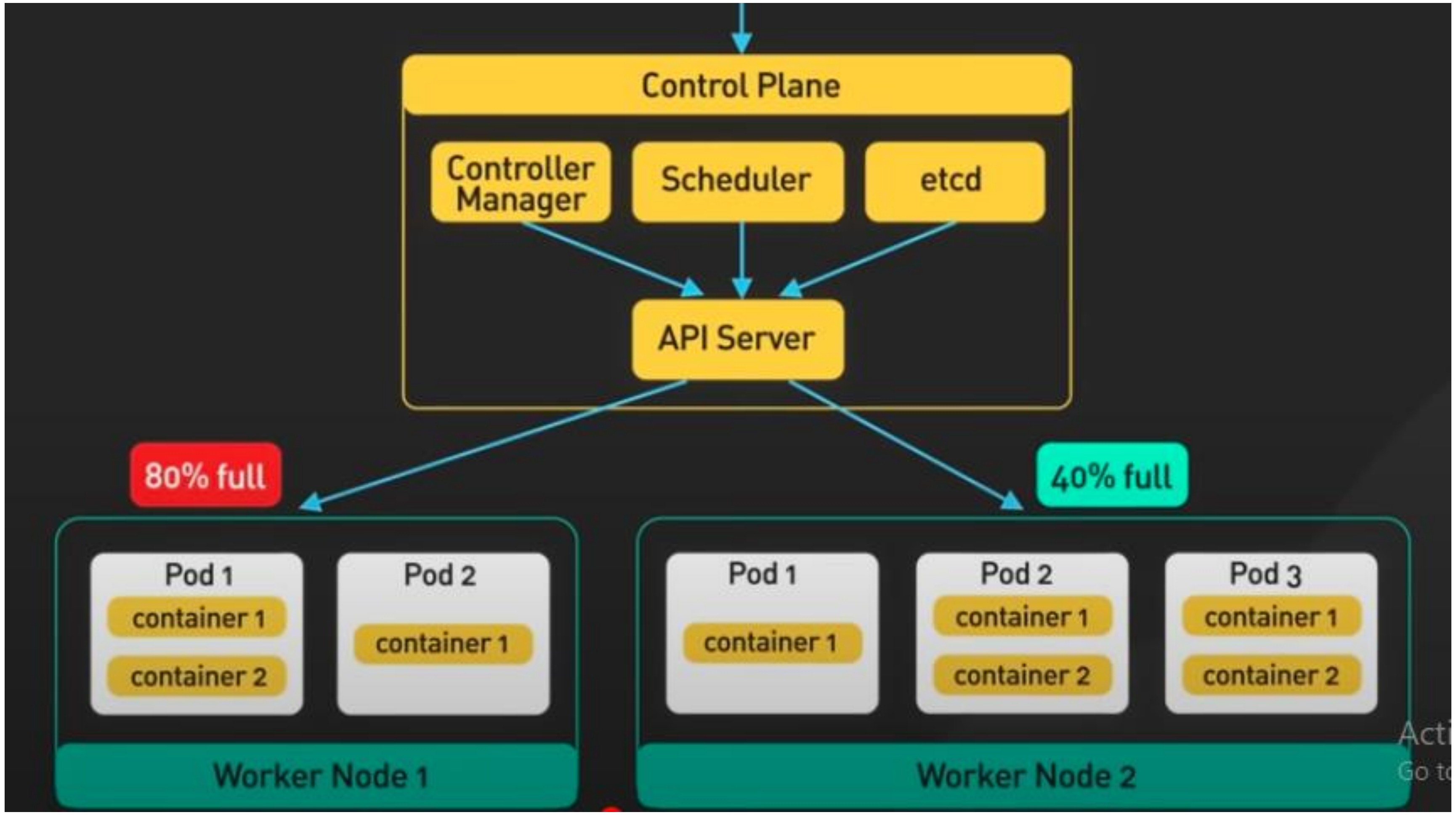

etcd
What states of the cluster are changed?
What resources are available?
Is the cluster healthy?
...

Scheduler

- A scheduler watches for newly created Pods that have no Node assigned. For every Pod that the scheduler discovers, the scheduler becomes responsible for finding the best Node for that Pod to run on.

- It uses the information the resources required by pods to make placement decision

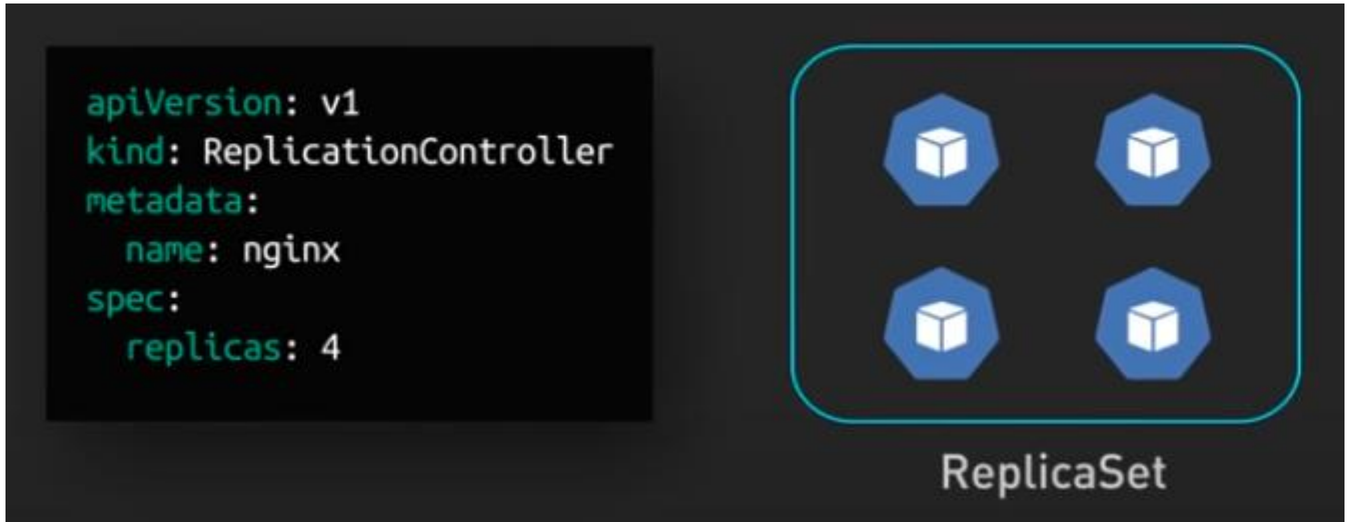
Scheduler


Control Plane
Controller Manager
Scheduler
etcd
API Server
80% full
40% full
Pod 1
container 1
container 2
Pod 2
container 1
Worker Node 1
Pod 1
container 1
Pod 2
container 1
container 2
Pod 3
container 1
container 2
Worker Node 2

- **Controllers in Kubernetes**
- A **Controller** is a control-loop that **watches the cluster state** (via the API Server) and makes changes to bring it closer to the **desired state** (stored in etcd).

Controllers

- **Examples of Controllers**
- **ReplicaSet Controller**
    - Ensures the correct number of Pod replicas are running.
    - Example: You say “I want 3 Pods.”
        - If only 2 Pods are running → controller creates 1 more.
        - If 4 are running → controller deletes 1.

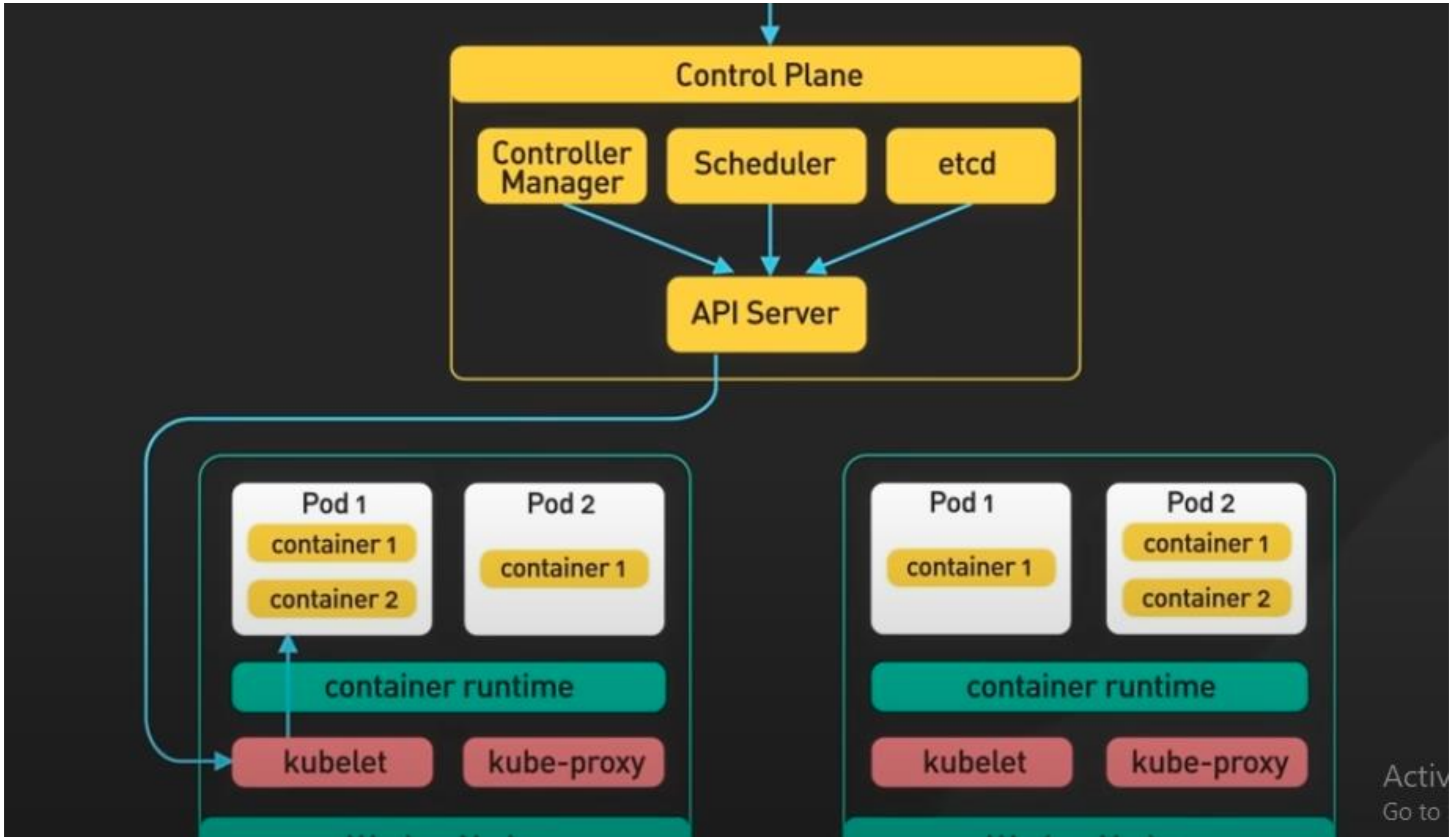


```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 4
```

kubelet

- The **kubelet** is an **agent** that runs on **every worker node** in a Kubernetes cluster.
- Its job is to **make sure containers are running in Pods as expected**.

kubelet


Control Plane
Controller Manager
Scheduler
etcd
API Server
Pod 1
container 1
container 2
Pod 2
container 1
container runtime
kubelet
kube-proxy
Pod 1
container 1
Pod 2
container 1
container 2
container runtime
kubelet
kube-proxy
Activ
Go to

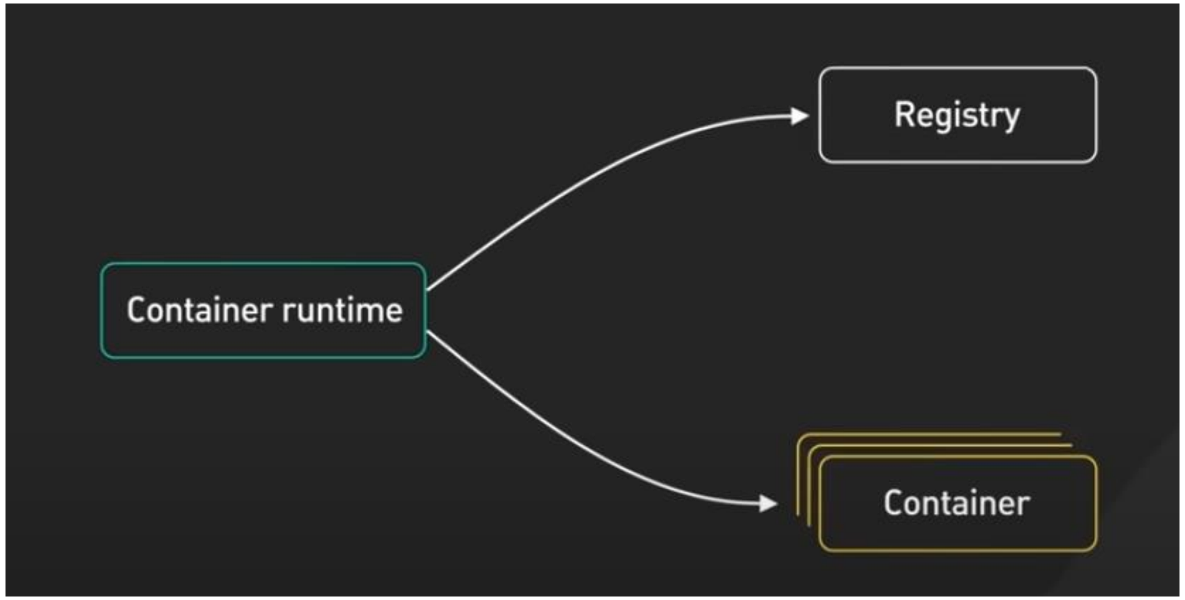
- **What kubelet does:**
- **Pod Lifecycle Manager**
  - Listens to the **API Server** for Pod specs assigned to its Node.
  - Starts, stops, and monitors containers (via the container runtime like Docker, containerd, or CRI-O).
- **Health Monitoring**
  - Performs **liveness/readiness probes** on containers.
  - Restarts containers if they fail.
- **Reports Node & Pod status**
  - Sends back status to the API Server (e.g., Node health, Pod running state).
- **Volume & Secrets Management**
  - Mounts **Volumes, ConfigMaps, Secrets** into Pods as needed.
- **Enforces desired state**
  - If the Pod spec says "run 2 containers," kubelet ensures they are running.
  - If someone manually kills a container → kubelet restarts it to match the spec.

Container runtime

- Runs the container in worker nodes
- A **Container Runtime** is the **software that actually runs containers** on a node.
- Kubernetes itself does **not run containers directly**.
- The **kubelet** talks to the container runtime to **start, stop, and manage containers**.

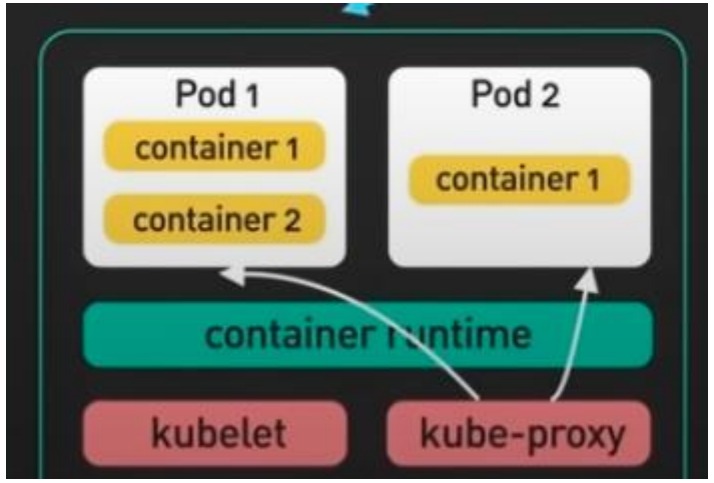
Container runtime

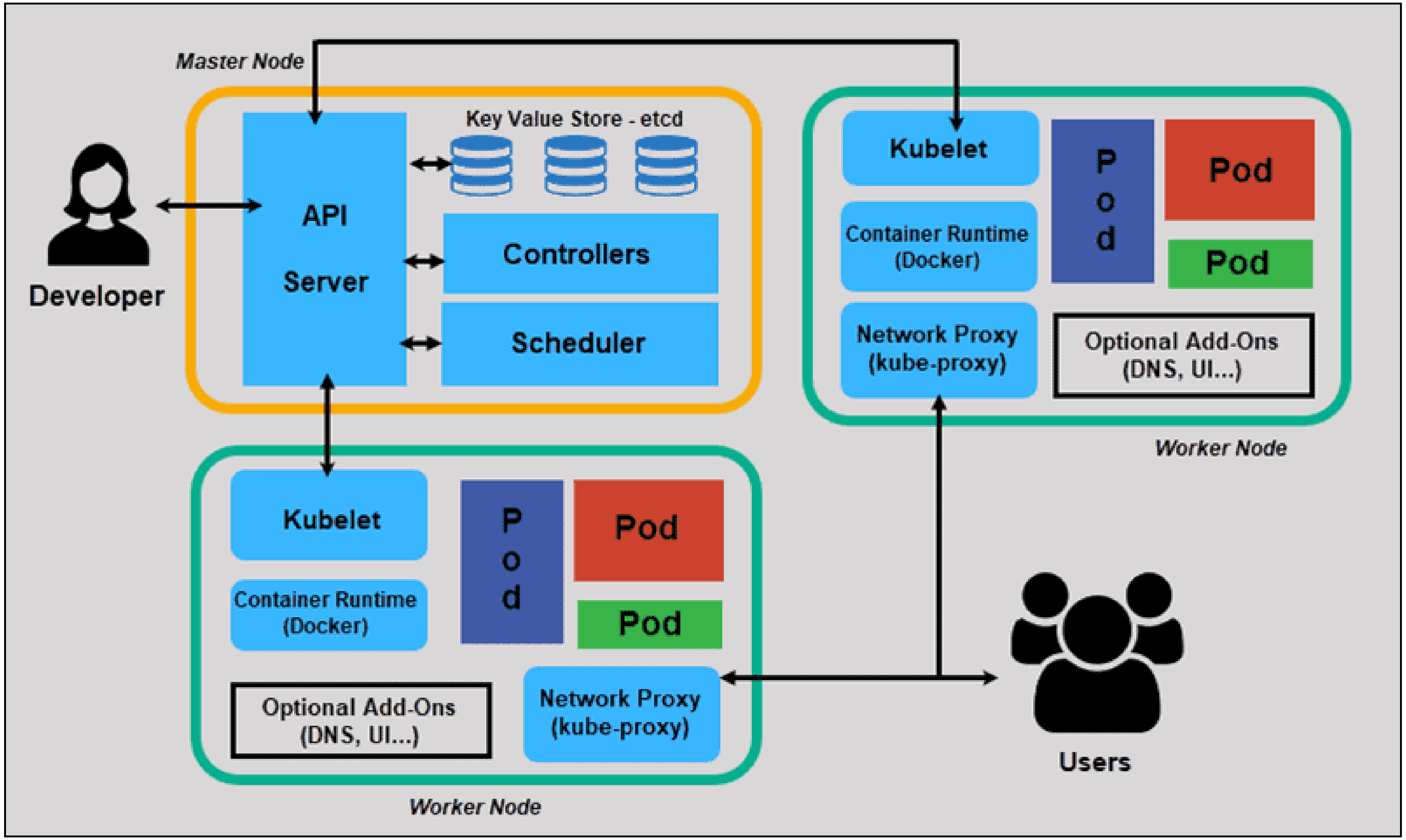
- A **container runtime** is responsible for **pulling container images from a registry** before running them.



kube-proxy

- **kube-proxy** is a networking proxy runs on **every worker node** and is responsible for **networking and traffic routing** inside the cluster.
- It also provides load balancing ensure traffic distributed




Master Node
Key Value Store - etcd
Developer
API Server
Controllers
Scheduler
Kubelet
Container Runtime (Docker)
Network Proxy (kube-proxy)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Worker Node
Kubelet
Container Runtime (Docker)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Network Proxy (kube-proxy)
Worker Node
Users