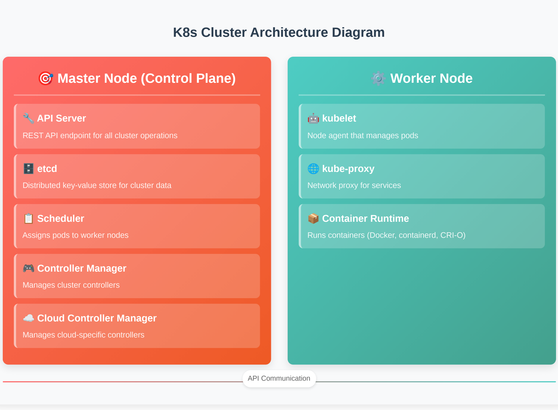
🚀 Kubernetes Cluster Architecture



🎯 Master Node Components (Control Plane)

🔧 API Server (kube-apiserver)

The **API Server** is the central management entity and the only component that directly interacts with etcd. It:

- Exposes the Kubernetes REST API
- Validates and processes API requests
- Authenticates and authorizes requests
- Acts as the frontend for the cluster's shared state

🗄️ etcd

A highly available **distributed key-value store** that:

- Stores all cluster data and configuration
- Maintains the desired state of the cluster
- Provides strong consistency and reliability
- Supports backup and restore operations

📋 Scheduler (kube-scheduler)

The **Scheduler** is responsible for:

- Selecting optimal nodes for pod placement
- Considering resource requirements and constraints
- Evaluating node affinity and anti-affinity rules
- Balancing workloads across the cluster

🎮 Controller Manager (kube-controller-manager)

Runs multiple **controllers** including:

- **Deployment Controller:** Manages ReplicaSets for deployments
- **ReplicaSet Controller:** Ensures desired number of pod replicas
- **Node Controller:** Monitors node health and status
- **Service Controller:** Manages service endpoints

☁️ Cloud Controller Manager (CCM)

Manages **cloud-specific controllers** for:

- Load balancers and external IPs
- Storage volumes and persistent volumes
- Node lifecycle management in cloud environments
- Integration with cloud provider APIs

⚙️ Worker Node Components

🤖 kubelet

The **node agent** that:

- Communicates with the API server
- Manages pods and containers on the node
- Reports node and pod status back to the control plane
- Performs health checks on containers
- Mounts volumes and handles pod networking

🌐 kube-proxy

A **network proxy** that:

- Implements Kubernetes service abstraction
- Manages network rules for service discovery
- Handles load balancing for services
- Maintains network connectivity between pods
- Supports different proxy modes (iptables, IPVS, userspace)

📦 Container Runtime

The **container runtime** options include:

- **Docker:** Most popular container runtime
- **containerd:** Lightweight, high-performance runtime
- **CRI-O:** OCI-compliant runtime specifically for Kubernetes
- **Responsibilities:** Pulling images, running containers, managing container lifecycle

🔄 How Components Work Together

Pod Creation Flow:

1. **User** submits a pod manifest to the **API Server**
2. **API Server** validates and stores the pod spec in **etcd**
3. **Scheduler** watches for unscheduled pods and assigns them to nodes
4. **kubelet** on the selected node pulls the pod spec and creates containers
5. **Container Runtime** pulls images and starts containers
6. **kube-proxy** updates network rules for the new pod

Service Discovery & Load Balancing:

- **Services** are created via API Server and stored in etcd
- **kube-proxy** watches for service changes and updates iptables rules
- **DNS** resolution is handled by CoreDNS (typically)
- **Load balancing** is performed by kube-proxy across healthy endpoints

🎯 Key Takeaways

- **Master Node:** Manages the cluster state and makes scheduling decisions
- **Worker Nodes:** Run the actual application workloads
- **API Server:** Central hub for all cluster communication
- **etcd:** Single source of truth for cluster state
- **kubelet:** Ensures pods are running as expected on each node
- **kube-proxy:** Handles network connectivity and service discovery