Kubernetes Architecture: Understanding the Core Components

This video provides a detailed explanation of Kubernetes architecture, comparing it to Docker to highlight the advantages and the need for each Kubernetes component.

Why Kubernetes?

Kubernetes offers four fundamental advantages over Docker:

- Cluster in Nature: Kubernetes is designed to operate as a cluster, unlike Docker which is typically a single-host solution.
- Auto-healing: It can automatically detect and recover from failures.
- Auto-scaling: Kubernetes can automatically adjust resources based on demand.
- Enterprise-level Support: It provides advanced features like load balancing, security, and networking.

Kubernetes Architecture Overview

Kubernetes architecture is primarily divided into two main planes:

- Control Plane (Master Component): Manages and controls the Kubernetes cluster.
- Data Plane (Worker Nodes): Runs the actual applications and workloads.

Data Plane Components (Worker Nodes)

The data plane components are responsible for running your applications.

1. Cubelet:
- Purpose: Responsible for running and maintaining pods on a node.
- Functionality: Ensures the pod is always running and informs the control plane if something goes wrong.
2. Container Runtime:
- Purpose: Provides the execution environment for containers within pods.
- Flexibility: Unlike Docker (which uses Docker shim), Kubernetes can use various container runtimes like `containerd`, `CRI-O`, or `Docker shim`, as long as they implement the Kubernetes Container Interface.
3. Kube-proxy:
- Purpose: Provides networking capabilities for pods.
- Functionality: Allocates IP addresses to pods, handles load balancing, and manages network rules using IP tables.

Control Plane Components (Master Component)

The control plane acts as the brain of the Kubernetes cluster, managing and orchestrating the data plane.

1. API Server:
- Purpose: The core component that exposes the Kubernetes API.
- Functionality: Receives all incoming requests from users and external systems, acting as the front end for the control plane.
2. Scheduler:
- Purpose: Responsible for scheduling pods and other resources onto available worker nodes.
- Functionality: Decides which node a new pod should run on, based on resource availability and other constraints.
3. etcd:
- Purpose: A distributed key-value store that serves as the backing store for all cluster data.
- Functionality: Stores the entire state and configuration of the Kubernetes cluster.
4. Controller Manager:
- Purpose: Manages various controllers that regulate the state of the cluster.
- Functionality: Ensures that the desired state of the cluster is maintained. For example, the `ReplicaSet` controller ensures a specified number of pod replicas are running.
5. Cloud Controller Manager (CCM):
- Purpose: Integrates Kubernetes with various cloud providers.
- Functionality: Allows Kubernetes to manage cloud-specific resources like load balancers and storage volumes. This component is only required when running Kubernetes on cloud platforms (e.g., EKS, AKS, GKE) and is not needed for on-premise deployments.