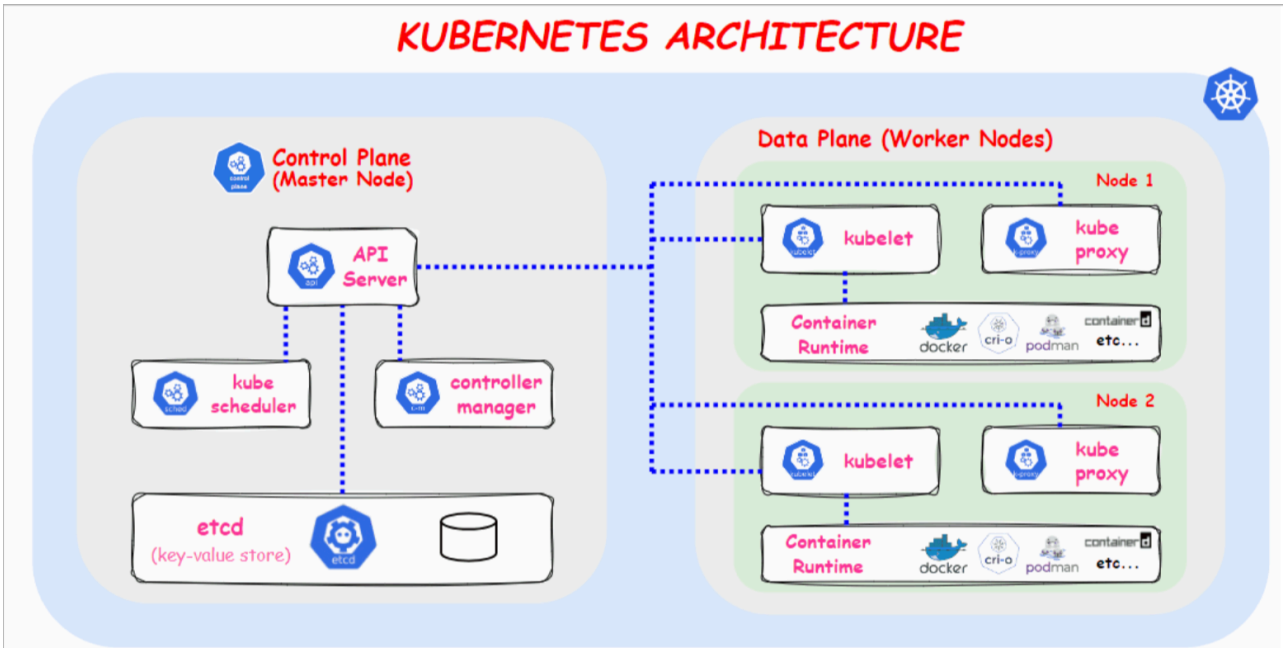
01 k8s arch

Kubernetes Architecture Guide



Hello World... Kubernetes has become a cornerstone in modern cloud-native infrastructure, revolutionizing the way organizations manage and scale their applications. As an open-source container orchestration platform, Kubernetes simplifies the deployment, scaling, and operation of containerized applications across clusters of machines. In today's fast-paced tech landscape, where agility, scalability, and efficiency are key, Kubernetes enables developers and businesses to build resilient, automated systems, fostering innovation and reducing operational complexity. Its widespread adoption has made it an essential tool for managing microservices architectures, hybrid clouds, and multi-cloud environments. So, let's get started with how the K8s architecture looks like...

Overview

Introduction

In a Kubernetes (K8s) cluster, the architecture is organized into two main components: the **control plane** and the **worker nodes**. Together, these components form a powerful system for managing containerized applications at scale.

Control Plane

The control plane is the brain of the Kubernetes cluster, acting as the central management hub for the entire infrastructure.

Key Responsibilities:

- Oversees the overall state and behavior of the cluster, ensuring everything is functioning as intended
- Maintains the desired state of the cluster by continuously comparing the current state with the desired state
- Takes corrective actions when discrepancies are detected (e.g., starting new pods or shutting down excess ones)
- Manages configurations for applications and services within the cluster

---

Components of the Control Plane

1. etcd

etcd is a reliable distributed key-value store that is simple, secure, and fast. It serves as Kubernetes' primary datastore, maintaining the current status, desired state, configuration, and metadata for all Kubernetes objects.

What information does etcd store?

- **Cluster Configuration and Metadata**: Nodes, Namespaces, Resource Quotas, Cluster Roles and Role Bindings (RBAC)
- **Workloads**: Pods, Deployments, ReplicaSets, DaemonSets, StatefulSets, Jobs, and CronJobs
- **Services and Networking**: Services, Endpoints, Ingress, Network Policies, and ConfigMaps
- **Secrets and Credentials**: Secrets and Service Accounts Tokens
- **Persistent Storage**: Persistent Volumes (PVs), Persistent Volume Claims (PVCs), and Storage Classes
- **Schedulers and Controllers**: Controllers, Events, Scheduler Info
- **Custom Resource Definitions (CRDs)**
- **Admission Controllers**: Admission Webhooks
- **Autoscaling Data**: Horizontal Pod Autoscalers (HPA)
- **API Server Configuration**: API Server Discovery Info
- **Federated Resources**

2. kube-api-server

The kube-api-server is a central component of the Kubernetes control plane, responsible for managing communication within the cluster. It serves as the primary management component, exposing a RESTful API that allows users and components to perform operations on Kubernetes resources like pods, services, and deployments.

Key Functionalities:

- RESTful Interface
- Resource Management (create, read, update, delete operations)
- Authentication and Authorization
- Admission Control
- Communication with etcd
- Event Notification
- Interaction with Controllers
- Error Handling

3. kube-controller-manager

The kube-controller-manager is a core component responsible for managing controllers that regulate the state of the cluster. Each controller is a separate process, but they are all run within a single binary called kube-controller-manager.

Key Characteristics:

- Continuously monitors the cluster state via the API server
- Makes decisions to bring the cluster closer to the desired state
- Supports high-availability setups with leader election
- Only one instance actively manages controllers at any given time

Common Controllers:

- Replication Controller
- Deployment Controller
- Node Controller
- Namespace Controller
- Ingress Controller
- And many more...

4. kube-scheduler

The kube-scheduler is a key component responsible for deciding which node an unscheduled pod will run on. It watches for newly created pods that have no assigned nodes and evaluates them against available nodes.

Scheduling Process:

1. When a pod is created, it enters the pending state if no nodes are available for scheduling
2. The scheduler filters out nodes that do not meet the pod's requirements (e.g., resource limits)
3. The remaining nodes are ranked based on various factors using a priority function (scores from 0 to 10)

Additional Features:

- Ensures efficient and optimal placement of pods
- Helps balance workloads and maintain resource availability
- Supports custom schedulers for implementing custom scheduling logic

---

Worker Nodes

Worker nodes, also known as minions, are where the actual applications run. Each worker node is managed by the control plane and runs the necessary services to execute and manage containerized applications.

Key Responsibilities:

- Provide necessary resources (CPU, memory, storage) for executing applications
- Provide isolation between different applications and workloads
- Implement security measures to protect running applications
- Ensure one workload does not interfere with another

Components of the Worker Node

1. kubelet

The kubelet is a crucial component responsible for managing the lifecycle of containers on individual nodes in a Kubernetes cluster. It runs on each node, ensuring that the containers described in PodSpecs are running as expected.

Key Functions:

- Constantly checks the health and status of pods and containers running on the node
- Restarts failed containers based on configuration
- Gathers resource utilization data (CPU, memory, etc.)
- Reports to the control plane for scheduling and scaling decisions
- Communicates with the container runtime to start, stop, and manage containers

2. Container Runtime Engine

The container runtime engine is a crucial component responsible for running and managing the containers.

Common Container Runtimes:

- Docker
- containerd
- CRI-O
- gVisor

Container Runtime Interface (CRI):

- An API in Kubernetes that allows Kubernetes to interact with different container runtimes
- Enables Kubernetes to support multiple container runtimes while maintaining a consistent interface

3. kube-proxy

kube-proxy is a critical component that manages network communication within a cluster. It runs as a process on each node in the cluster.

Key Functions:

- Manages a pod network (internal virtual network extending across all nodes)
- Sets up necessary routing rules when a Service is created
- Ensures traffic sent to a Service's cluster IP is appropriately routed to backing Pods
- Handles load balancing across multiple Pod instances

Example: Restaurant Analogy

To better understand Kubernetes architecture, imagine a restaurant where various roles and operations come together to provide a smooth dining experience. The **Kubernetes cluster** functions like a restaurant, with its management and staff working together to deliver services efficiently.

The Restaurant-Kubernetes Mapping

Restaurant Component	Kubernetes Component	Role
Restaurant	Kubernetes Cluster	The entire system where operations work together
Restaurant Management	Master Node (Control Plane)	Central management system overseeing all operations
Front Desk	API Server	First point of contact; validates and distributes requests
Restaurant Manager	Controller Manager	Monitors workflow and maintains desired state
Kitchen Dispatcher	Scheduler	Assigns tasks based on workload and capacity
Restaurant Details	etcd	Tracks all details (cash, staff, dishes, supplies)
Restaurant Staff	Worker Nodes	Carry out actual tasks
Chef	Kubelet	Follows instructions and prepares/manages workloads
Waitstaff	Kube-Proxy	Delivers services to the right destination
Kitchen	Container Runtime	Where the actual work happens
Dishes	Pods	Composed of different elements working together
Menu	Services	Provides stable endpoints for accessing pods
Restaurant Sections	Namespaces	Organizes space for different environments

Detailed Analogies

1. Kubernetes Cluster = Restaurant The restaurant represents the whole Kubernetes cluster, a system where various operations (like seating, cooking, and serving) work together to provide a seamless experience.

2. Master Node (Control Plane) = Restaurant Management This central management system oversees the entire restaurant's operations, just like the control plane in Kubernetes manages the cluster.

3. API Server = Front Desk

- *Role in the Restaurant:* The front desk is the first point of contact, where customer orders are taken and directed to the right place
- *Role in Kubernetes:* The API server is the main entry point to the Kubernetes cluster, validating, processing, and distributing requests

4. Controller Manager = Restaurant Manager

- *Role in the Restaurant:* The restaurant manager monitors the workflow, ensuring enough food is prepared, and adjusting staffing levels as needed
- *Role in Kubernetes:* The controller manager monitors the cluster and maintains the desired state

5. Scheduler = Kitchen Dispatcher

- *Role in the Restaurant:* The dispatcher assigns orders to chefs based on the complexity of dishes and the chef's current workload
- *Role in Kubernetes:* The scheduler assigns new workloads (pods) to the most appropriate nodes

6. Etcd = Restaurant Details

- *Role in the Restaurant:* Tracks the cash details, staff details, dishes details, supplies details and all
- *Role in Kubernetes:* Etcd stores the cluster's configuration and state

7. Worker Nodes = Restaurant Staff Each worker node represents the restaurant's staff who carry out tasks like preparing food and serving customers.

8. Kubelet = Chef

- *Role in the Restaurant:* The chef follows instructions from the front desk and prepares dishes according to the orders
- *Role in Kubernetes:* Kubelet ensures that containers assigned to the worker node are running

9. Kube-Proxy = Waitstaff

- *Role in the Restaurant:* The waitstaff ensures that food is delivered to the right table
- *Role in Kubernetes:* Kube-Proxy routes network traffic to the correct pods

10. Container Runtime = Kitchen

- *Role in the Restaurant:* The kitchen is where the ingredients are transformed into dishes
- *Role in Kubernetes:* The container runtime runs containers on the worker nodes

11. Pods = Dishes

- *Role in the Restaurant:* Each dish is composed of different elements (e.g., burger = bun + patty + toppings)
- *Role in Kubernetes:* Pods are the smallest deployable units containing one or more containers

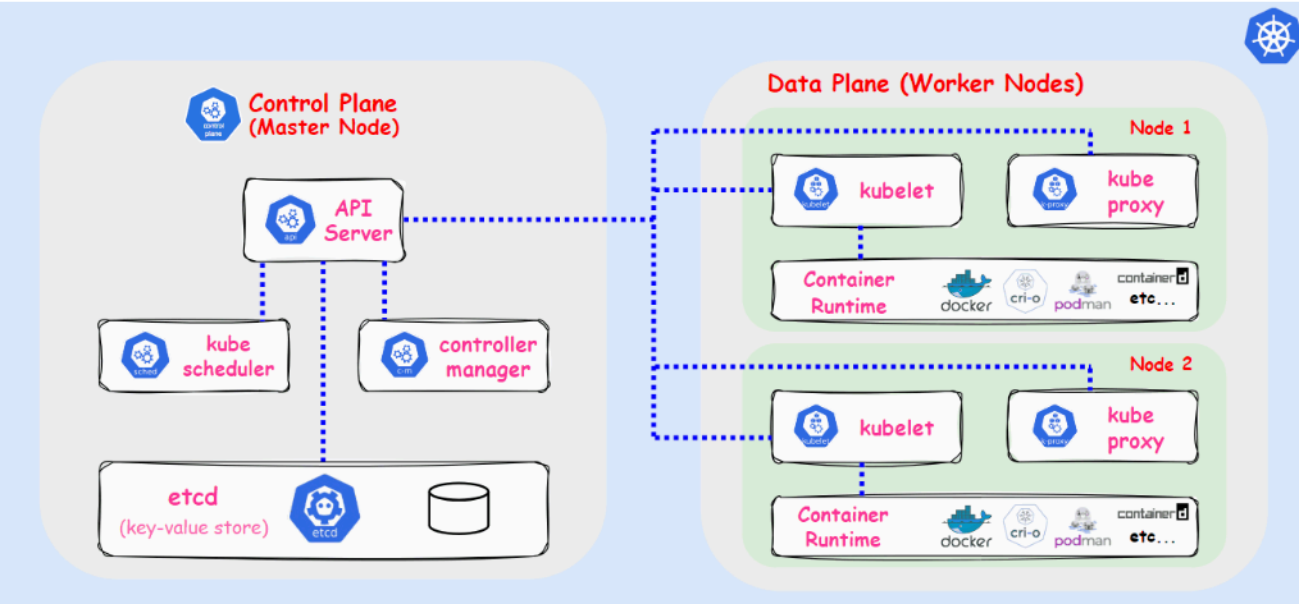
12. Services = Menu

- *Role in the Restaurant:* The menu provides customers with a list of available dishes
- *Role in Kubernetes:* Services abstract a group of pods, providing stable endpoints

13. Namespaces = Restaurant Sections

- *Role in the Restaurant:* Different sections (family area, bar, outdoor patio) organize the space
- *Role in Kubernetes:* Namespaces create separate environments (development, production)

Architecture Diagram



Tags

#kubernetes #k8s #architecture #containerization #cloud-native #devops