Kubernetes Architecture

Training Material

Table of Contents

Chapter 1: Introduction to Kubernetes

1.1 What is Kubernetes?

Kubernetes, often abbreviated as K8s, is an open-source platform designed to automate the deployment, scaling, and management of containerized applications. It provides a robust and flexible framework for orchestrating containerized workloads across clusters of machines. By abstracting away the complexities of infrastructure, Kubernetes empowers developers to focus on building and deploying applications without worrying about the underlying hardware.

1.2 History of Kubernetes

Kubernetes originated at Google as a project to manage large-scale containerized workloads. It was open-sourced in 2014 and quickly gained popularity due to its ability to simplify container orchestration. The Cloud Native Computing Foundation (CNCF) now oversees the development and stewardship of Kubernetes.

1.3 Why Kubernetes?

There are numerous reasons why Kubernetes has become the de facto standard for container orchestration:

- **Scalability:** Kubernetes can handle both small and large-scale deployments, automatically scaling applications up or down based on demand.
- **Reliability:** It ensures high availability and fault tolerance by automatically restarting failed containers and replicating workloads across multiple nodes.
- **Efficiency:** Kubernetes optimizes resource utilization by efficiently packing containers onto nodes.
- **Portability:** Applications deployed on Kubernetes can run consistently across different cloud providers and on-premises infrastructure.
- **Developer Productivity:** Kubernetes simplifies the development and deployment process, allowing developers to focus on writing code rather than managing infrastructure.

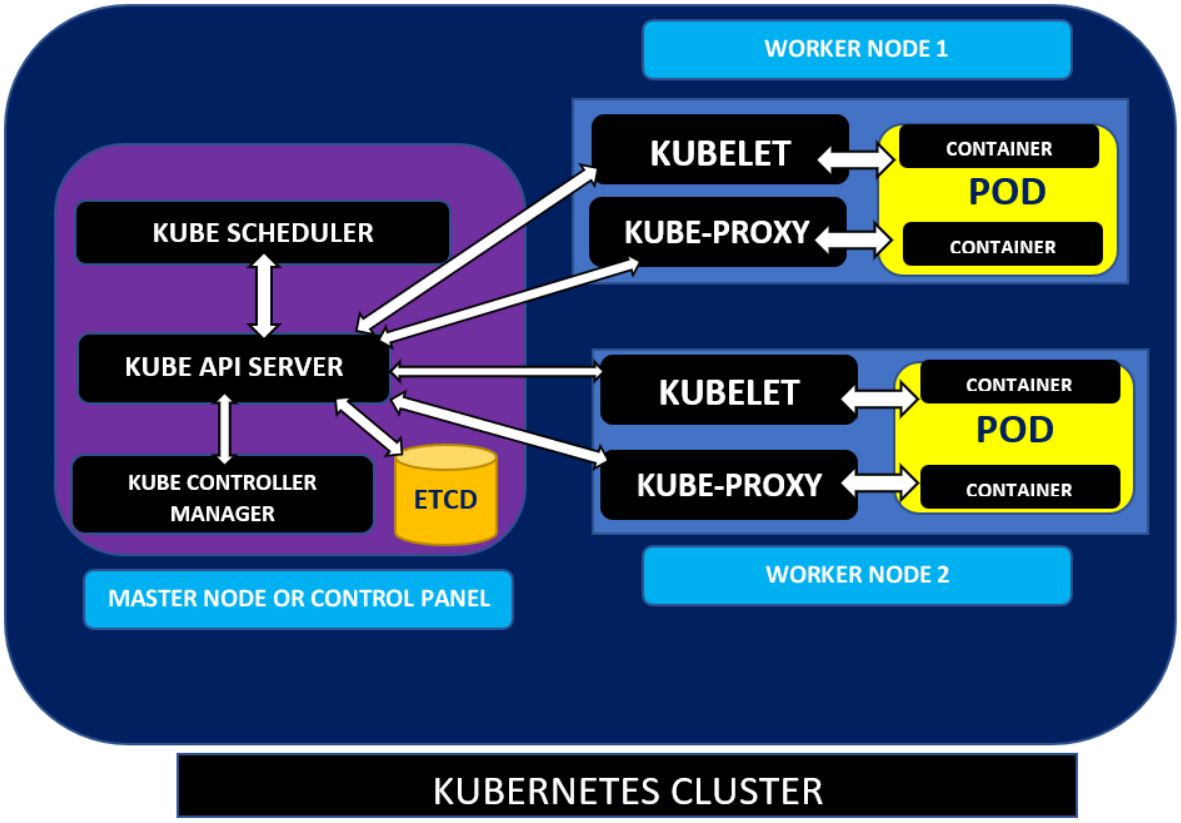
1.4 Key Concepts and Terminology

To understand Kubernetes, it's essential to grasp some key concepts and terminology:

- **Cluster:** A group of machines (nodes) that work together to run containerized applications.
- **Node:** A physical or virtual machine that hosts containers.
- **Pod:** The smallest deployable unit of computing, consisting of one or more containers with shared storage and network resources.
- **Service:** An abstraction layer that provides stable network identities for pods.
- **Namespace:** A mechanism for isolating groups of resources within a cluster.

1.5 Kubernetes Architecture Overview

The Kubernetes architecture consists of several key components:



- **Control Plane:**
  - **etcd:** A distributed, consistent key-value store that stores the cluster state.
  - **kube-apiserver:** The front-end component that exposes the Kubernetes API.
  - **kube-scheduler:** Assigns Pods to Nodes.
  - **kube-controller-manager:** Ensures the state of the cluster matches the user-specified state.
- **Node:**
  - **kubelet:** Manages the Node and communicates with the Control Plane.
  - **kube-proxy:** Implements network proxy functionality.
  - **Container Runtime:** Executes container images (e.g., Docker, containerd, CRI-O).

Chapter 2: Kubernetes Core Components

2.1 Node

A Kubernetes cluster consists of one or more nodes. Each node is a physical or virtual machine that runs containerized applications. Nodes can be classified into two types:

Master Node: The master node is responsible for managing the entire cluster. It includes the following components:

- **etcd:** A distributed, consistent key-value store that stores the cluster's configuration and state.
- **kube-apiserver:** The front-end component that exposes the Kubernetes API.
- **kube-scheduler:** Assigns Pods to Nodes.
- **kube-controller-manager:** Ensures the state of the cluster matches the user-specified state.

Worker Node: Worker nodes execute Pods. They run the following components:

- **kubelet:** Manages the Node and communicates with the Control Plane.
- **kube-proxy:** Implements network proxy functionality.
- **Container Runtime:** Executes container images (e.g., Docker, containerd, CRI-O).

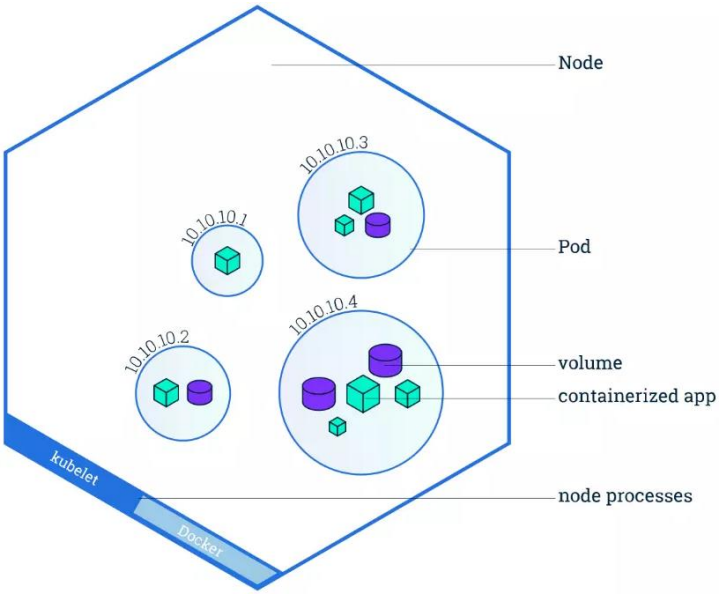
2.2 Pod

A Pod is the smallest deployable unit of computing in Kubernetes. It consists of one or more containers, sharing storage and network resources.

Pod Structure:

A Pod typically includes:

- **Containers:** One or more containers that share the same network namespace and storage volume.
- **Volume:** A directory that can be mounted by containers within the Pod.
- **Network:** A network interface that allows containers within the Pod to communicate with each other and with the outside world.



Pod Lifecycle:

A Pod goes through the following lifecycle stages:

1. **Pending:** The Pod is scheduled but not yet running.
2. **Running:** The Pod is running on a Node.
3. **Succeeded:** The Pod has finished executing successfully.
4. **Failed:** The Pod has failed to execute.

Pod Specifications:

Pod specifications are defined in YAML or JSON files. They include information such as:

- Container images
- Resource requests and limits
- Environment variables
- Volume mounts
- Network configuration

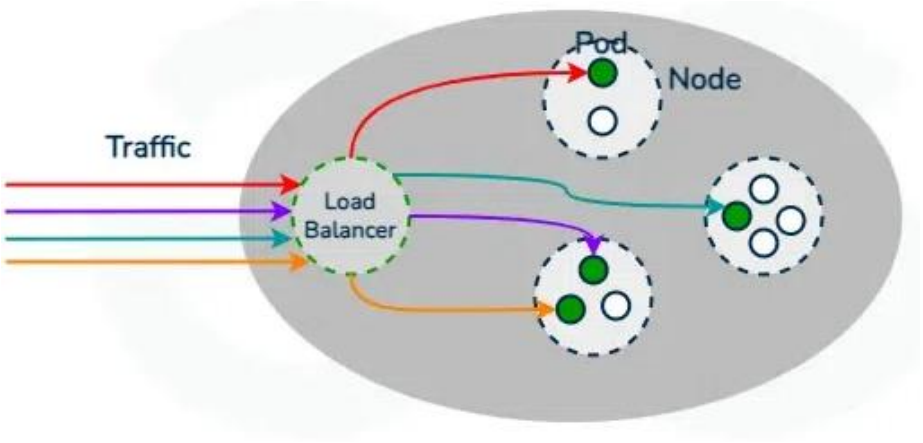
2.3 Service

A Service provides a stable network identity for a set of Pods. It enables communication between Pods, regardless of their IP addresses or hostnames.

Service Types:

Kubernetes supports three types of Services:

- **ClusterIP:** A ClusterIP Service is only accessible within the cluster.
- **NodePort:** A NodePort Service exposes the Service on each Node's IP address at a static port.
- **LoadBalancer:** A LoadBalancer Service exposes the Service externally using a cloud provider's load balancer.



Service Discovery:

Kubernetes provides a built-in service discovery mechanism that allows Pods to find and communicate with each other. This is achieved through DNS and labels.

Service Mesh:

A Service Mesh is a layer of infrastructure that provides advanced networking capabilities for microservices. It handles tasks such as service discovery, load balancing, security, and observability. Popular Service Mesh solutions include Istio and Linkerd.

2.4 Namespace

A Namespace is a mechanism for isolating groups of resources within a cluster. It provides a way to organize and manage resources, such as Pods, Services, and Deployments, into logical groups.

Namespace Isolation:

Namespaces provide isolation between different teams or projects. Resources in one namespace cannot access resources in another namespace, unless explicitly authorized.

Resource Quotas and Limits:

Namespaces can be configured with resource quotas and limits to control the amount of resources that can be used by Pods within the namespace.

Namespace Creation and Deletion:

Namespaces can be created and deleted using the kubectl command-line tool.

Chapter 3: Hands-on Lab: Setting up a Kubernetes Cluster and Deploying Applications

3.1 Setting up a Kubernetes Cluster

3.1.1 Using Minikube

Minikube is a tool that lets you run a single-node Kubernetes cluster on your local machine. This is a great way to get started with Kubernetes without needing a cloud provider.

Steps:

1. **Install Minikube:** Follow the instructions for your operating system on the Minikube website.
2. **Start Minikube:** Run the minikube start command.
3. **Configure kubectl:** Run the minikube config command to configure your kubectl to point to the Minikube cluster.

3.1.2 Using a Cloud Provider

Cloud providers like AWS, GCP, and Azure offer managed Kubernetes services that simplify the setup and management of Kubernetes clusters.

Steps:

1. **Create a Cloud Account:** Sign up for an account with your preferred cloud provider.
2. **Create a Kubernetes Cluster:** Use the cloud provider's console or CLI to create a Kubernetes cluster.
3. **Configure kubectl:** Configure your kubectl to point to your cloud provider's cluster.

3.2 Deploying Applications

3.2.1 Creating YAML Files

Kubernetes uses YAML files to define the desired state of your applications. A YAML file typically defines:

- **Pods:** The containers that make up your application.
- **Services:** The network endpoints for your application.
- **Deployments:** The desired state of your application.

Example YAML file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container

        image: my-app-image:latest
```

```
    ports:
    - containerPort:
8080
```

3.2.2 Using kubectl to Deploy

The kubectl command-line tool is used to interact with Kubernetes clusters. To deploy an application, you can use the kubectl apply command:

kubectl apply -f my-app.yaml

3.2.3 Verifying Deployments

To verify that your application has been deployed successfully, you can use the following kubectl commands:

- **Get Pods:** kubectl get pods
- **Get Services:** kubectl get services
- **Get Deployments:** kubectl get deployments

You can also use kubectl describe to get more detailed information about a specific resource. For example:

kubectl describe pod my-app-pod

By following these steps, you can set up a Kubernetes cluster and deploy your applications.

Sources and related content