What is Kubernetes? and Basic Components of Kubernetes Architecture

Table of Contents

What is Kubernetes?

Kubernetes is a powerful open-source orchestration tool.

Designed to help you manage microservices and containerized applications across a distributed cluster of computing nodes.

Kubernetes lets you create, deploy, manage, and scale application containers across one or more host clusters.

Kubernetes aims to hide the complexity of managing containers through the use of several key capabilities, such as REST APIs and declarative templates that can manage the entire lifecycle.

Kubernetes was originally developed by Google. It is now supported by the Cloud Native Computing Foundation (CNCF). Kubernetes is portable and can run on any public or private cloud platform, including AWS, Azure, Google Cloud, and OpenStack, as well as on bare metal machines.

Features

- A highly resilient infrastructure.
- Zero downtime deployment.
- Automated rollback, self-healing, and scaling of containers.
- Automated self healing capabilities, such as auto-restart, auto-placement, and auto-replication.

Use Cases

- Schedule and run containers on clusters of either physical or virtual machines (VMs).
- Fully implement and utilize a container-based infrastructure in your production environments.
- Automate operational tasks.
- Create cloud-native applications with Kubernetes as a runtime platform

Advantages

- Orchestrate your containers across several hosts.
- Optimize hardware usage to make the most of your resources.
- Mount and add storage to run stateful apps.
- Automate and control application deployments and updates.
- Scale your containerized applications as well as their resources—on the fly.
- Declaratively manage services to ensure deployed applications run as intended.
- Perform health checks and enable application self-healing.

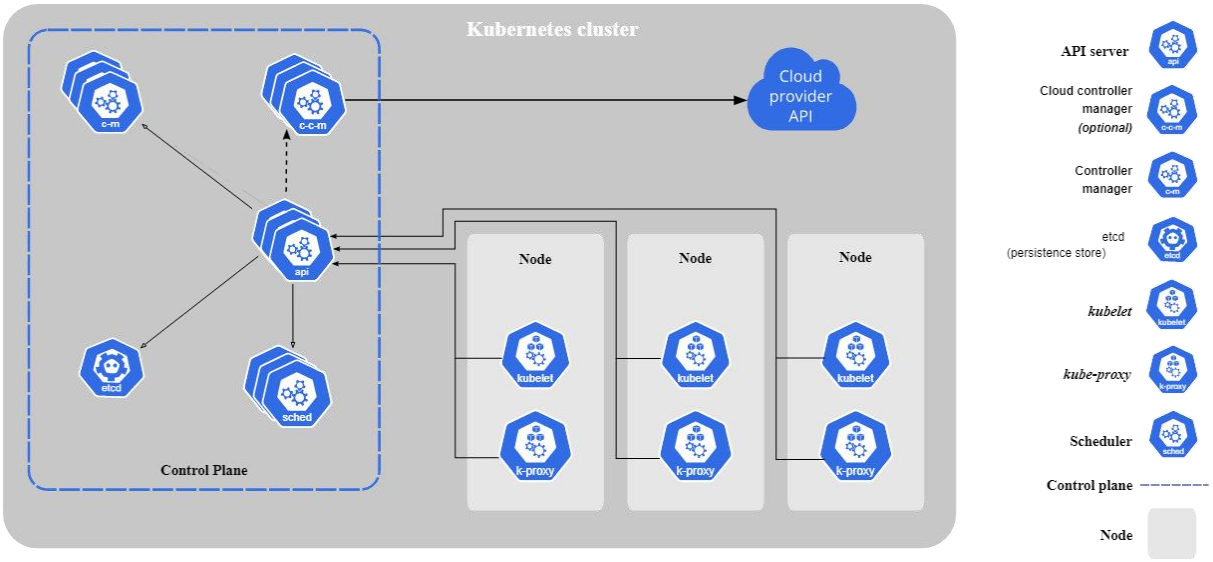
Kubernetes Architecture

Kubernetes architecture is, at its foundation, a client-server architecture.

The server side of Kubernetes is the known as the control plane. By default, there is a single control plane server that acts as a controlling node and point of contact. This server consists of components including the kube-apiserver, etcd storage, kube-controller-manager, cloud-controller-manager, kube-scheduler, and Kubernetes DNS server.

The client side of Kubernetes comprises the cluster nodes—these are machines on which Kubernetes can run containers. Node components include the kubelet and kube-proxy. Can be virtual machines (VMs) or physical machines. A node hosts pods, which run one or more containers.

- Service Side - Master node
- Client Side - Worker node



Ref: Kubernetes

Environments running Kubernetes consist of the following key components:

- Kubernetes control plane (Master)—manages Kubernetes clusters and the workloads running on them. Include components like the API Server, Scheduler, and Controller Manager.
- Kubernetes data plane (Worker)—machines that can run containerized workloads. Each node is managed by the kubelet, an agent that receives commands from the control plane.

- Pods—pods are the smallest unit provided by Kubernetes to manage containerized workloads. A pod typically includes several containers, which together form a functional unit or microservice.
- Persistent storage—local storage on Kubernetes nodes is ephemeral, and is deleted when a pod shuts down. This can make it difficult to run stateful applications. Kubernetes provides the Persistent Volumes (PV) mechanism, allowing containerized applications to store data beyond the lifetime of a pod or node.

Kubernetes Core Components | Kubernetes Nodes

- Control Plane | Master Node
- Worker Nodes

Control Plane | Master Node Components

A control plane serves as a nerve center of each Kubernetes cluster. It includes components that can control your cluster, its state data, and its configuration.

The Kubernetes control plane is responsible for ensuring that the Kubernetes cluster attains a desired state, defined by the user in a declarative manner. The control plane interacts with individual cluster nodes using the kubelet, an agent deployed on each node.

Components:

- Kube-apiserver
- Kube-scheduler
- Kube-controller-manager
- Etcd
- Cloud-controller-manager

Components in Details:

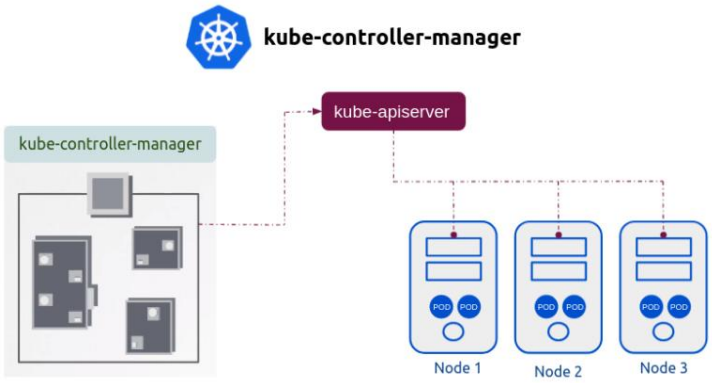
- Kube-apiserver
  - Provides an API that serves as the front end of a Kubernetes control plane.
  - It is responsible for handling external and internal requests—determining whether a request is valid and then processing it.
  - The API can be accessed via the kubectl command-line interface or other tools like kubeadm, and via REST calls.

- This is the only component that communicates with the etcd cluster, making sure data is stored in etcd and is in agreement with the service details of the deployed pods.

- Kube-scheduler
  - This component is responsible for scheduling pods on specific nodes according to automated workflows and user defined conditions, which can include resource requests, concerns like affinity and taints or tolerations, priority, persistent volumes (PV), and more.
  - It reads the service's operational requirements and schedules it on the best fit node.
  - It reads the service's operational requirements and schedules it on the best fit node.
  - For example, if the application needs 1GB of memory and 2 CPU cores, then the pods for that application will be scheduled on a node with at least those resources. The scheduler runs each time there is a need to schedule pods. The scheduler must know the total resources available as well as resources allocated to existing workloads on each node.

- Kube-controller-manager
  - It manages various controllers in Kubernetes. Controllers are control loops that continuously watch the state of your cluster, then make or request changes where needed. Each controller tries to move the current cluster state closer to the desired state.
  - Controllers continuously talk to the kube-apiserver and the kube-apiserver receives all information of nodes through Kubelet.

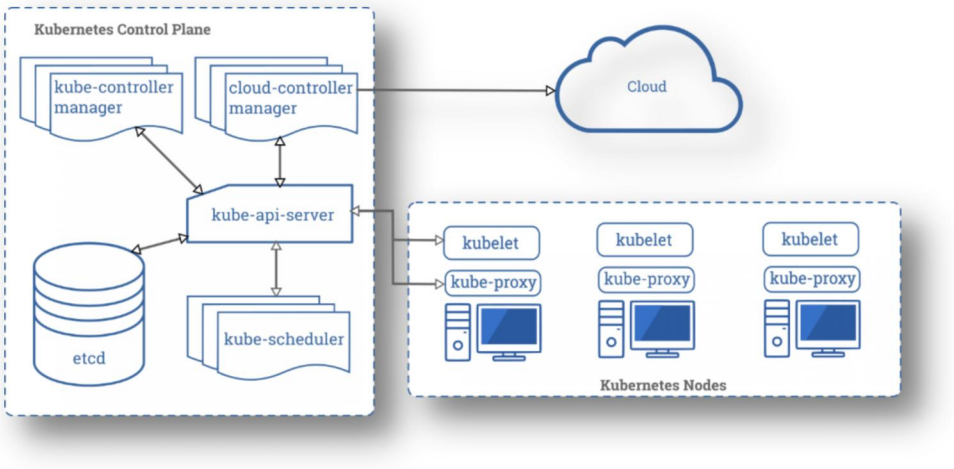
  - Types of controllers
    - Node controller: Responsible for noticing and responding when nodes go down.
    - Replication controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
    - Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
    - Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.



Ref: blog.knoldus.com

- Etcd
  - A key-value database that contains data about your cluster state and configuration. Etcd is fault tolerant and distributed. (such as the number of pods, their state, namespace, etc.).
  - It should only be accessible from the API server for security reasons. etcd enables notifications to the cluster about configuration changes with the help of watchers. Notifications are API requests on each etcd cluster node to trigger the update of information in the node's storage.
- Cloud-controller-manager
  - A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster.
  - This cloud-controller-manager runs only controllers specific to the cloud provider. It is not required for on-premises Kubernetes environments. It uses multiple, yet logically-independent, control loops that are combined into one binary, which can run as a single process. It can be used to add scale a cluster by adding more nodes on cloud VMs, and leverage cloud provider high availability and load balancing capabilities to improve resilience and performance.
  - The cloud-controller-manager only runs controllers that are specific to your cloud provider. If you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.
  - The following controllers can have cloud provider dependencies:
    - Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
    - Route controller: For setting up routes in the underlying cloud infrastructure

- Service controller: For creating, updating and deleting cloud provider load balancers



Ref: kubernetes-docsy-staging.netlify.app

Worker Nodes Components

Worker nodes within the Kubernetes cluster are used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated. The worker nodes perform any actions triggered via the Kubernetes API, which runs on the master node.

A master node is a node which controls and manages a set of worker nodes (workloads runtime) and resembles a cluster in Kubernetes.

Components:

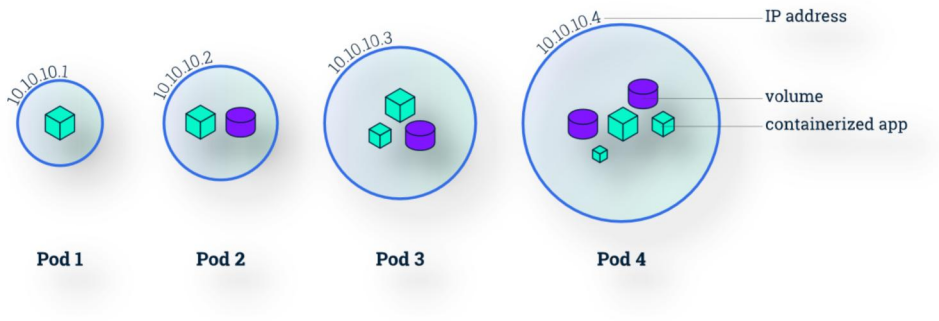
- Kubelet
- Kube-proxy
- Pods
- Nodes
- Container networking
- Container runtime engine

Components in Details:

- Kubelet
  - Kubelet main service on a node, which manages the container runtime (such as containerd or CRI-O). The kubelet regularly takes in new or modified pod

specifications (primarily through the kube-apiserver) and ensures that pods and their containers are healthy and running in the desired state. This component also reports to the master on the health of the host where it is running.

- Kube-proxy
  - A proxy service that runs on each worker node to deal with individual host subnetting and expose services to the external world. It performs request forwarding to the correct pods/containers across the various isolated networks in a cluster.
- Pods
  - A pod is the smallest unit of management in a Kubernetes cluster.
  - It represents one or more containers that constitute a functional component of an application. Pods encapsulate containers, storage resources, unique network IDs, and other configurations defining how containers should run.
  - A container runs logically in a pod (though it also uses a container runtime); A group of pods, related or unrelated, run on a cluster. A pod is a unit of replication on a cluster; A cluster can contain many pods, related or unrelated [and] grouped under the tight logical borders called namespaces.



Ref: ngugijoan.medium.com

- Nodes
  - Nodes are physical or virtual machines that can run pods as part of a Kubernetes cluster. A cluster can scale up to 5000 nodes. To scale a cluster's capacity, you can add more nodes.
- Container Networking
  - Container networking enables containers to communicate with hosts or other containers. It is often achieved by using the container networking interface (CNI), which is a joint initiative by Kubernetes, Apache Mesos, Cloud Foundry, Red Hat OpenShift, and others.

    - CNI offers a standardized, minimal specification for network connectivity in containers. You can use the CNI plugin by passing the kubelet --network-plugin=cni command-line option. The kubelet can then read files from --cni-conf-dir and use the CNI configuration when setting up networking for each pod.

- Container Runtime Engine
    - A container runtime, also known as container engine, is a software component that can run containers on a host operating system. In a containerized architecture, container runtimes are responsible for loading container images from a repository, monitoring local system resources, isolating system resources for use of a container, and managing container lifecycle.
    - Common container runtimes commonly work together with container orchestrators. The orchestrator is responsible for managing clusters of containers, taking care of concerns like container scalability, networking, and security. The container engine takes responsibility for managing the individual containers running on every compute node in the cluster.

Ref:

- https://kubernetes.io/
- https://techmormo.com
- https://ngugijoan.medium.com/
- https://blog.knoldus.com/
- https://www.aquasec.com/
- https://www.suse.com/
- https://docs.oracle.com/
- https://komodor.com/