Kubernetes Architecture and Components Explained

Itay Gershon
Product Manager, Intel Granulate

What is Kubernetes Architecture?

Kubernetes is a popular open-source platform for managing and deploying containerized applications. It was originally developed by Google and is now maintained by the Cloud Native Computing Foundation. Kubernetes provides a way to automate the deployment, scaling, and management of containerized applications, allowing developers to focus on writing code instead of worrying about the underlying infrastructure.

Kubernetes has a highly modular architecture that allows it to be flexible and scalable. At a high level, it is made up of a number of components, including the Kubernetes master, the API server, etcd, the scheduler, the controller manager, the kubelet, and the Kubernetes networking component. These components work together to provide a platform for managing and deploying containerized applications in a cluster environment.

In this article:

Kubernetes Architecture and Components Explained
Control Plane Components

Kubernetes Architecture and Components Explained

The Kubernetes architecture consists of various components that help manage clusters. A Kubernetes cluster is a group of machines, called nodes, that are used to run containerized applications. These nodes are managed by the Kubernetes platform, which provides a way to automate the deployment, scaling, and management of the applications.

Pods are the basic units of deployment in Kubernetes. A pod is a group of one or more containers that are deployed together on the same node. Pods are used to host the applications that run on the cluster, and they provide a way to manage the containers as a single entity.

This is part of an extensive series of guides about DevOps.

Image Source: Kubernetes

Control Plane Components

The Kubernetes control plane is the central management point for the cluster, and it consists of a number of different components that work together to manage the cluster and ensure that the desired state of the system is maintained.

kube-apiserver

The kube-apiserver is the central management point for a Kubernetes cluster. It is one of the key components of the Kubernetes control plane, and it exposes the Kubernetes API, which is used by external clients to interact with the cluster.

The kube-apiserver is responsible for coordinating the various components of the cluster, and it provides the mechanisms for enforcing policies and ensuring that the desired state of the cluster is maintained. It also provides an authentication and authorization layer for the cluster, ensuring that only authorized users and applications are able to access the API.

kube-scheduler

This control plane process makes scheduling decisions while considering collective and individual resource requirements, affinity and anti-affinity specifications, inter-workload interference, data locality, deadlines, and hardware, software, or policy constraints.

kube-controller-manager

The kube-controller-manager is a component of the Kubernetes control plane that is responsible for managing the various controllers that handle tasks such as replicating containers and ensuring that the desired state of the cluster is maintained.

Some types controllers that are managed by the kube-controller-manager include:

- **Deployment controllers:** Deployment controllers manage the deployment of a group of replicas of a containerized application. They are responsible for ensuring that the desired number of replicas are running and available, and they can be used to perform rolling updates of the application.
- **Replication controllers:** Replication controllers ensure that a specified number of replicas of a pod (a group of one or more containers) are running at any given time. If a pod fails or is terminated, the replication controller will create a new one to replace it.
- **StatefulSet controllers:** StatefulSet controllers are similar to Deployment controllers, but they are designed specifically for managing stateful applications. They provide features such as persistent storage, unique network identities, and ordered deployment and scaling.
- **DaemonSet controllers:** DaemonSet controllers ensure that a specific pod is running on every node in the cluster, or on a subset of nodes that match a certain label. They are commonly used for deploying pods that perform cluster-wide tasks, such as logging or monitoring.

cloud-controller-manager

It is responsible for managing the cloud-specific controllers that handle tasks such as creating and deleting cloud resources, and it provides a consistent interface for the other components of Kubernetes to interact with the cloud infrastructure.

Node Components

In Kubernetes, nodes are the worker machines in the cluster. They play a crucial role in the functioning of the system, providing the computing resources that are needed to run the applications and hosting the pods that contain the applications.

kubelet

The kubelet is a process that runs on each node in a Kubernetes cluster. It is responsible for managing the containers on the node, including starting and stopping containers, as well as monitoring their health and taking action if necessary.

The kubelet uses PodSpecs to determine what containers should be run on the node and how they should be configured. This allows the kubelet to manage the containers on the node and ensure that they are running properly.

kube-proxy

The kube-proxy runs on each node in the cluster, and it is responsible for managing the network connectivity for the containers on that node. It uses the network policies that are specified for the cluster to determine how traffic should be handled, and it implements these policies by configuring the network rules on the node.

Container runtime

A container runtime is the software that is responsible for running containers on a computer. It is the component that actually starts and stops the containers, and it is responsible for managing the resources that are allocated to the containers. There are several different container runtime options that can be used with Kubernetes, such as containerd.

Kubernetes Services

Service. When you create a Service, you specify the labels that the Service should match, and the Service will automatically discover and proxy traffic to the matching pods.

There are several types of Services in Kubernetes, each with its own set of characteristics and uses. The most common types of Services are:

ClusterIP: This type of Service is only accessible within the cluster and is not exposed to the outside world. It is useful for exposing services to other parts of the cluster, such as for load balancing or for accessing other services within the cluster.

NodePort: This type of Service exposes a specific port on each node in the cluster, allowing you to access the Service from outside the cluster using the node's IP address and the specified port.

LoadBalancer: This type of Service creates a load balancer in the cloud provider's infrastructure, allowing you to access the Service from the outside world using a public IP address.

ExternalName: This type of Service allows you to access an external service using a DNS name, rather than exposing it directly through Kubernetes.

You can use Services to expose your applications to other parts of the cluster or to the outside world, to load balance traffic between multiple replicas of your application, and to abstract away the details of the underlying pods, allowing you to make changes to your application without having to update any references to it.

Kubernetes Networking

In Kubernetes, the networking model for cluster-wide, pod-to-pod networking is based on the concept of a virtual network, which provides connectivity between all pods in the cluster. This virtual network is implemented using the Container Network Interface (CNI), which is a plugin-

cluster. Pods can communicate with each other using this IP address, regardless of which node they are running on.

The CNI is responsible for managing the virtual network and ensuring that all pods are correctly connected to it. When a pod is created, the CNI is responsible for allocating an IP address for the pod and configuring the necessary networking resources, such as virtual network interfaces and routing tables.

The CNI uses a variety of plugins to provide networking for pods. Some common plugins include Flannel, Calico, and Weave Net.

You can use the CNI and its plugins to configure the networking for your Kubernetes cluster and to ensure that all pods are correctly connected to the virtual network. This allows you to easily deploy and manage applications in a scalable and reliable way.

Kubernetes Persistent Storage

In Kubernetes, persistent storage is a way to store data in containers in a way that it persists even if the container is stopped or deleted. This is implemented using:

PersistentVolumes (PVs)

PVs are the underlying storage resources that are available in a Kubernetes cluster. They represent actual storage resources, such as a block storage device, that can be used to store data. PVs are created and managed by cluster administrators, and are independent of any individual pod or container.

PersistentVolumeClaims (PVCs)

PVCs are requests for storage made by individual pods or containers. They specify the amount and type of storage that a pod or container needs, and are then matched to available PVs by the Kubernetes system. Once a PVC has been bound to a PV, the pod or container can use the storage provided by the PV to store data in a way that persists even if the pod or container is deleted.