

[Become a Certified Professional →](#)

Understanding Kubernetes Architecture

Last updated on May 20,2020 23.8K Views



Containers have become the definitive way to develop applications because they provide packages that contain everything you need to run your applications. In this blog, we will discuss Kubernetes architecture and the moving parts of **Kubernetes** and also what are the key elements, what are the roles and responsibilities of them in Kubernetes architecture.

Kubernetes: An Overview

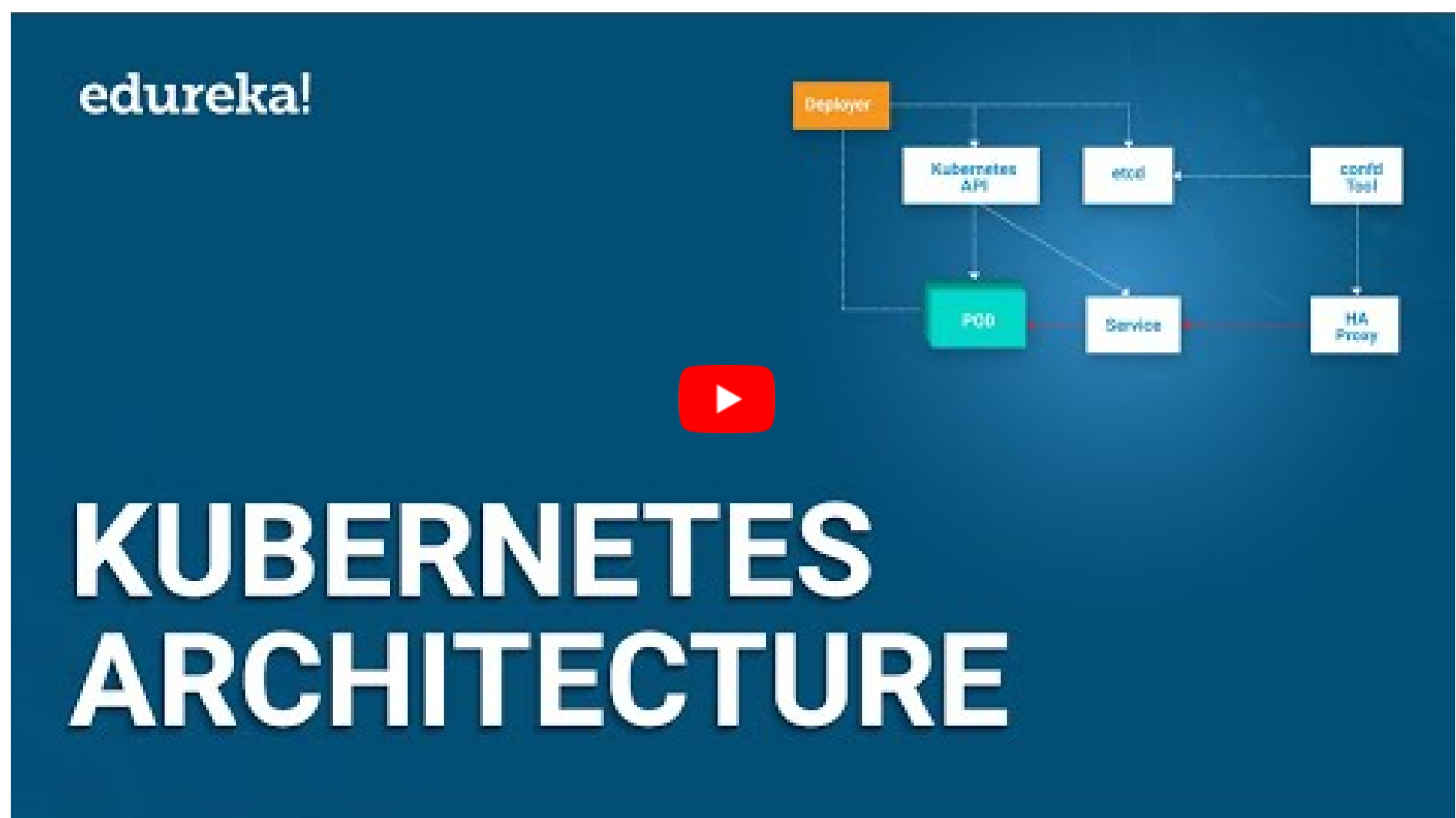
Kubernetes is an open-source Container Management tool which automates container deployment, container (de)scaling & container load balancing.

- Written on Golang, it has a huge community because it was first developed by Google & later donated to **CNCF**
- Can group 'n' no of containers into one logical unit for managing & deploying them

If you would favor a video explanation on Kubernetes Architecture, then you can go through the below video.



Kubernetes Architecture | Edureka



This video will give you an introduction to popular DevOps tool – Kubernetes, and will deep dive into Kubernetes Architecture and its working.

Want To Explore More About Kubernetes

[Explore Now](#)

Features Of Kubernetes

For a detailed explanation, check this [blog](#).



FREE WEBINAR

Build Continuous Delivery Pipeline i...

Become a Certified Professional →



Kubernetes Architecture/Kubernetes Components



 FREE WEBINAR



Build Continuous Delivery Pipeline i...

Become a Certified Professional→



Kubernetes Architecture has the following main components:

- Master nodes
- Worker/Slave nodes
- Distributed key-value store(etcd.)

Master Node

It is the entry point for all administrative tasks which is responsible for managing the Kubernetes cluster. There can be more than one master node in the cluster to check for fault tolerance. More than one master node puts the system in a High Availability mode, in which one of them will be the main node which we perform all the tasks.

For managing the cluster state, it uses **etcd** in which all the master nodes connect to it.

Let us discuss the components of a master node. As you can see in the diagram it consists of 4 components:

API server:



Become a Certified Professional→

- *Instructor-led Sessions*
- *Assessments*
- *Lifetime Access*
- *24 x 7 Expert Support*

Explore Curriculum

- Performs all the administrative tasks through the API server within the master node.
- In this REST commands are sent to the API server which validates and processes the requests.
- After requesting, the resulting state of the cluster is stored in the distributed key-value store.

Scheduler:

- The scheduler schedules the tasks to slave nodes. It stores the resource usage information for each slave node.
- It schedules the work in the form of Pods and Services.
- Before scheduling the task, the scheduler also takes into account the quality of the service requirements, data locality, affinity, anti-affinity, etc.

Controller manager:

- Also known as **controllers**.
- It is a daemon which regulates the Kubernetes cluster which manages the different non-terminating control loops.
- It also performs lifecycle functions such as namespace creation and lifecycle, event garbage collection, terminated-pod garbage collection, cascading-deletion garbage collection, node garbage collection, etc.
- Basically, a controller watches the desired state of the objects it manages and watches their current state through the API server. If the current state of the objects it manages does not meet the desired state, then the control loop takes corrective steps to make sure that the current state is the same as the desired state.

What is the ETCD?

- etcd is a distributed key-value store which stores the cluster state.
- It can be part of the Kubernetes Master, or, it can be configured externally.
- etcd is written in the Go programming language. In Kubernetes, besides storing the cluster state (based on the **Raft Consensus Algorithm**) it is also used to store configuration details such as subnets, ConfigMaps, Secrets, etc.
- A raft is a consensus algorithm designed as an alternative to Paxos. The Consensus problem involves multiple servers agreeing on values; a common problem that arises in the context of replicated state machines. Raft defines three different roles (Leader, Follower, and Candidate) and achieves consensus via an elected leader

Now you have understood the functioning of Master node. Let's see what is the Worker/Minions node and its components.

Worker Node (formerly minions)

It is a physical server or you can say a VM which runs the applications using Pods (**a pod scheduling unit**) which is controlled by the master node. On a physical server (worker/slave node), pods are scheduled. For accessing the applications from the external world, we connect to nodes.

Let's see what are the following components:

Container runtime:

- To run and manage a container's lifecycle, we need a **container runtime** on the worker node.
- Sometimes, Docker is also referred to as a container runtime, but to be precise, Docker is a platform which uses **containers** as a container runtime.

Kubelet:

- It is an agent which communicates with the Master node and executes on nodes or the worker nodes. It gets the Pod specifications through the API server and executes the containers associated with the Pod and ensures that the containers described in those Pod are running and healthy.

Kube-proxy:



FREE WEBINAR

Build Continuous Delivery Pipeline i...




Become a Certified Professional➔



DEVOPS
CERTIFICATION
TRAINING

DevOps Certification Training


Reviews
★★★★★ 5(86871)



AWS CERTIFIED
DEVOPS ENGINEER
TRAINING

AWS Certified DevOps Engineer Training

Reviews
★★★★★ 5(3586)



KUBERNETES
CERTIFICATION
TRAINING COURSE

Kubernetes Certification Training Course

Reviews
★★★★★ 5(6328)



DOCKER CERTIFIED
ASSOCIATE TRAINING

Docker Certified Associate Training

Reviews
★★★★★ 5(5443)

- Kube-proxy runs on each node to deal with individual host sub-netting and ensure that the services are available to external parties.
- It serves as a network proxy and a load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets.
- It is the network proxy which runs on each worker node and listens to the API server for each Service endpoint creation/deletion.
- For each Service endpoint, kube-proxy sets up the routes so that it can reach to it.

Pods

A pod is one or more containers that logically go together. Pods run on nodes. Pods run together as a logical unit. So they have the same shared content. They all share the same IP address but can reach other Pods via localhost, as well as shared storage. Pods don't need to all run on the same machine as containers can span more than one machine. One node can run multiple pods.

Use Case: How Luminis Technologies used Kubernetes in production

Problem: Luminis, a software technology company used AWS for deploying their applications. For deploying the applications, it required custom scripts and tools to automate which was not easy for teams other than operations. Their small teams didn't have the resources to learn all of the details about the scripts and tools.

Main Issue: There was no **unit-of-deployment** which created a gap between the development and the operations teams.

Solution:

How did they Deploy in Kubernetes:

Become a Certified Professional →




They used a [blue-green deployment](#) mechanism to reduce the complexity of handling multiple concurrent versions. (As there's always only one version of the application running in the background)

In this, a component called “**Deployer**” that orchestrated the deployment was created by their team by open sourcing their implementation under the Apache License as part of the Amdatu umbrella project. This mechanism performed the health checking on the pods before re-configuring the load balancer because they wanted each component that was deployed to provide a health check.

How did they Automate Deployments?

With the **Deployer** in place, they were able to engage up deployments to a build pipeline. After a successful build, their build server pushed a new Docker image to a registry on Docker Hub. Then the build server invoked the **Deployer** to automatically deploy the new version to a test environment. That same image was promoted to production by triggering the **Deployer** on the production environment.



[Kubernetes Certification Training Course](#)

[Weekday / Weekend Batches](#)

See Batch Details

 FREE WEBINAR

Build Continuous Delivery Pipeline i...

^