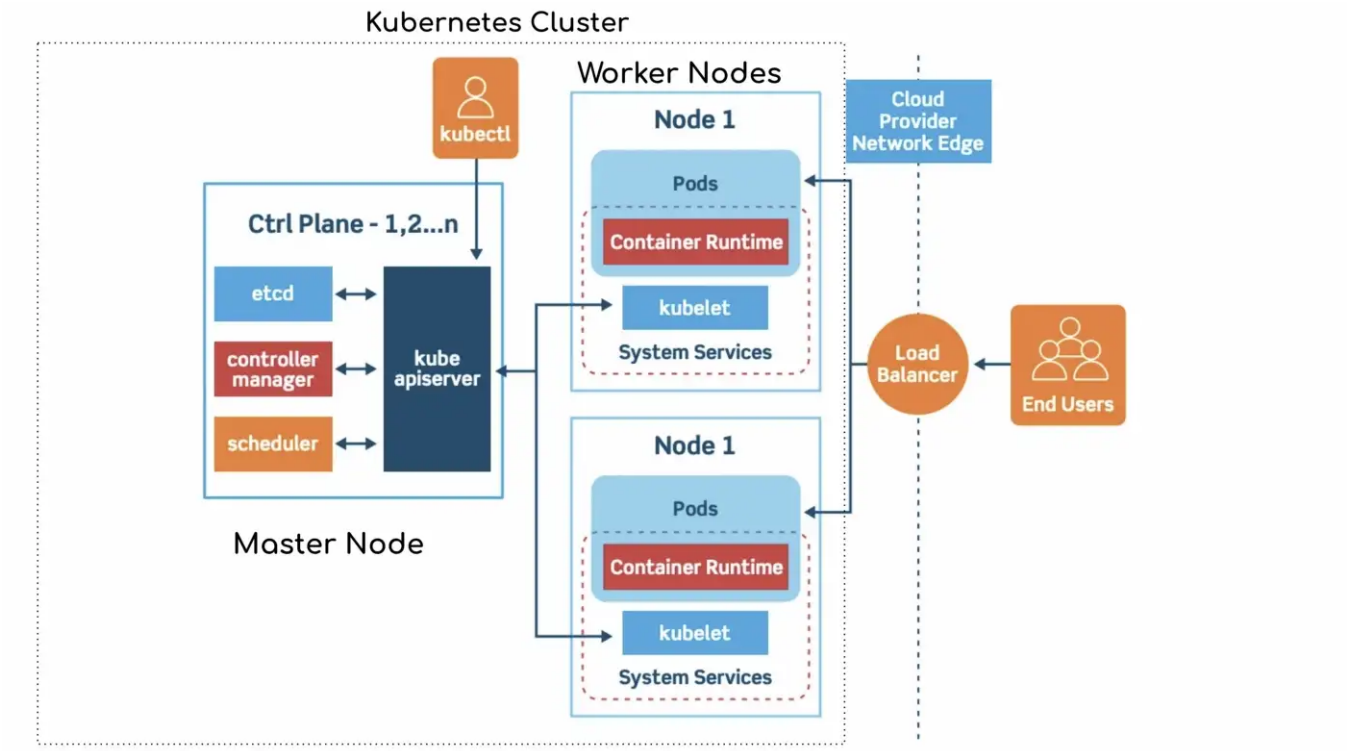
Published in SysopsMicro

@pramodAIML Follow
Jan 3, 2021 · 8 min read · Listen

Kubernetes Fundamentals For Absolute Beginners: Architecture & Components

Learning Kubernetes architecture & components



source:

> *The very need to make our complex application highly available, scalable, portable, and deployable in small modules independently lead to the birth of Kubernetes*

Today we will cover :

- What Is Kubernetes?

- Why Kubernetes?
- Kubernetes Architecture
- Key Components of Kubernetes

What Is Kubernetes?

Kubernetes is popularly known as K8s

K8s is a production-grade open-source container orchestration tool developed by Google to help you manage the containerized/dockerized applications supporting multiple deployment environments like On-premise, cloud, or virtual machines.

> *k8s automates the deployment of your containerised images and helps the same to scale horizontally to support high level of application availability*

Why K8s: What Problem Does K8S Solve?

One of the primary reasons why K8s became so popular is the ever-growing demand for businesses to support their micro service-driven architectural needs.

Microservice architecture helps companies in :

- Independently develop and deploy their complex applications by breaking them into small scalable modules
- Help them work in multiple small teams supporting individual application modules, to be developed and deployed with the required speed and agility

This desire of the companies to move from legacy monolith to Microservices has led to the creation of large containerized applications. Each container image in itself is a Microservice that needs to be managed and scaled efficiently with less overhead, this demand to handle thousands and thousands of container became a tedious task for the organization. This problem led to the evolution of K8s as one of the popular container orchestration tools.

The organization adopted a container orchestration tool like Kubernetes due to the following key benefits

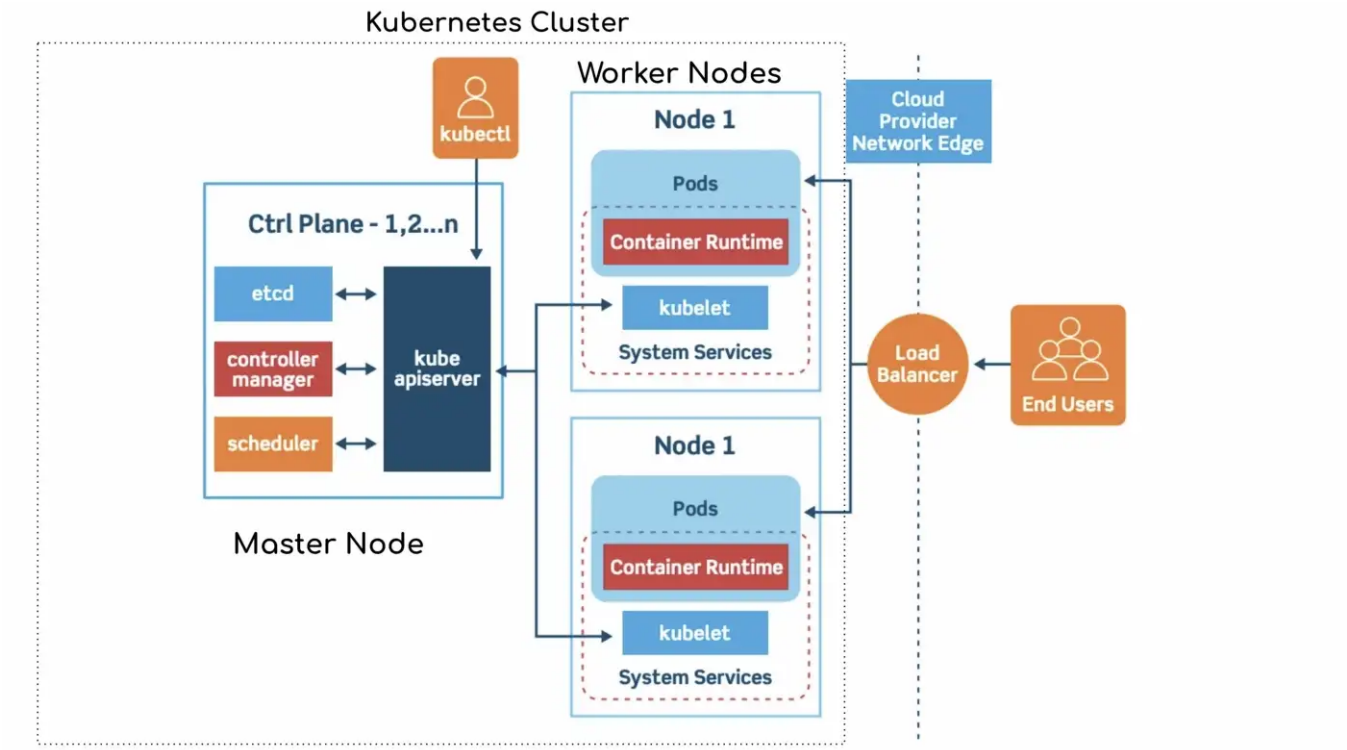
What features Does K8s Offer?

- Assures **high availability** with zero downtime

- Highly performant and **scalable**
- Reliable infrastructure to support **data recovery** with ease

Now that we have understood why K8s is imperative, it's time to decode the underlying architecture of K8s

Fundamental Architecture Of Kubernetes Cluster:



Kubernetes cluster's most basic architecture has two major Nodes

- **Master Nodes**
- **Worker Nodes or Slave Nodes**

If one follows the official documentation of Kubernetes, it becomes extremely overwhelming to grab the concepts for them. So we will try to comprehend the same with required simplification.

Let's understand first how does worker nodes or slave nodes in K8s works and what are the key component of the worker nodes

Worker Node In K8s Cluster:

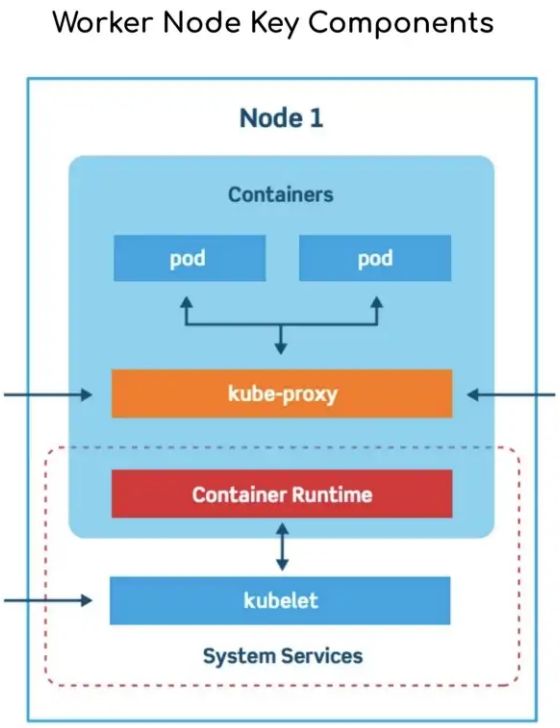


Fig: 2.0: Worker Node Components in K8s cluster

As a developer or K8s administrator most of the time you will deal with worker nodes, whether you have to deploy your containerized app or you have to autoscale it, or you have to roll out any new app update on your production-grade server, you will often deal with worker nodes.

As this node does the actual work required by the cluster administrator or developer, it is known as worker nodes. Worker node can have one or more pods, these pods are your abstraction of a containerized application. Every worker as shown in the fig:2.0 runs these 3 key processes

- **Container Runtime**
- **kubelet**
- **kube-proxy**

Container Runtime:

Every Microservice module(micro-app) you deploy is packaged into a single pod that has its own container runtime. One needs to install a container runtime into each worker node in the cluster so that Pods can run there.

Some of the container runtime examples are,

- **containerd**

- **CRI-O**
- **Docker**

kubelet:

kubelet is a primary node-agent of the worker node, which interacts with both node and the container in the given worker node.

The **kubelet** is responsible for

- Maintaining a set of pods, which are composed of one or more containers, on a local system.
- For registering a node with a Kubernetes cluster, sending events and pod status, and reporting resource utilization.

Within a Kubernetes cluster, the **kubelet** watches for **PodSpecs** via the Kubernetes API server.

A PodSpec is a YAML or JSON object that describes a pod. The **kubelet** takes a set of PodSpecs that are provided through various mechanisms (primarily through the **API server**) and ensures that the containers described in those PodSpecs are running and healthy.

> *The Kubelet is the primary and most important controller in Kubernetes. It's responsible for driving the container execution layer, typically Docker.*

Kube-proxy:

K8s cluster can have multiple worker nodes and each node has multiple pods running, so if one has to access this pod, they can do so via Kube-proxy.

> *kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.*

In order to access the pod via k8s services, there are certain network policies, that allow network communication to your Pods from network sessions inside or outside of your cluster. These rules are handled via **kube-proxy**

kube-proxy has an intelligent algorithm to forward network traffics required for pod access which minimized the overhead and makes service communication more performant

So far we have seen that these 3 processes need to be installed and running successfully within your worker nodes in-order to manage your containerized application efficiently, but the bigger question is

- *Who manages these worker nodes, to ensure that they are always up and running?*
- *How does the K8s cluster know which pods should be scheduled and which one should be dropped or restarted?*
- *How does the k8s cluster know the resource level requirements of each container app?*

Well the answer lies in the concept of Master Node, let's explore the same below

Master Node in K8s cluster:

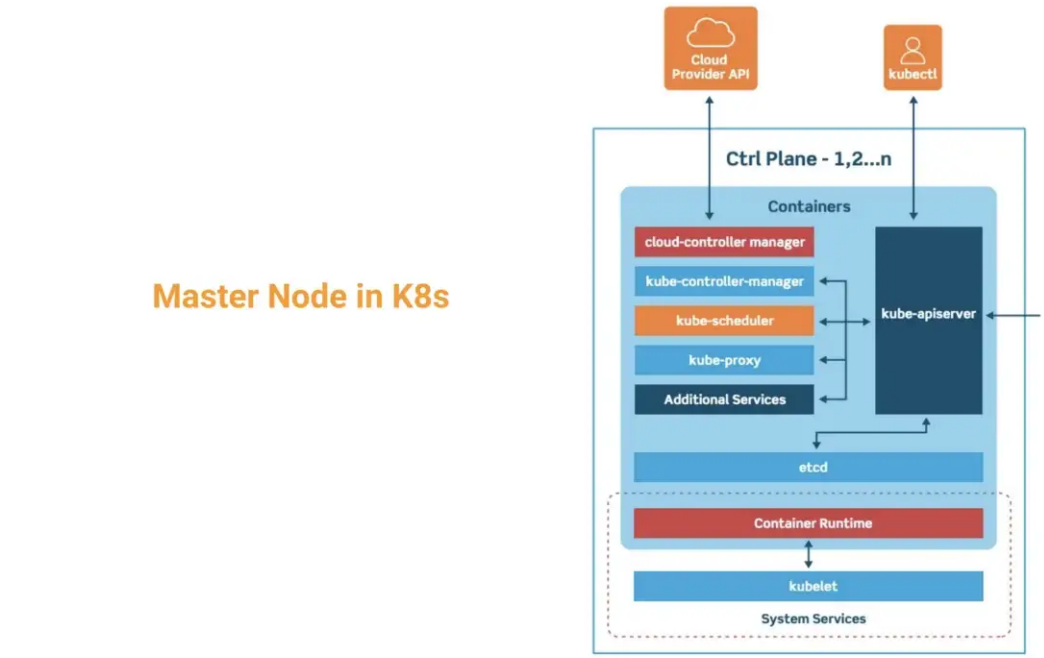


Fig:3.0 Master Node Processes in K8s

The **master node** is also known as a **control plane** that is responsible to manage worker/slave nodes efficiently. They interact with the worker node to

- Schedule the pods
- Monitor the worker nodes/Pods
- Start/restart the pods

- Manage the new worker nodes joining the cluster

Master Node Processes:

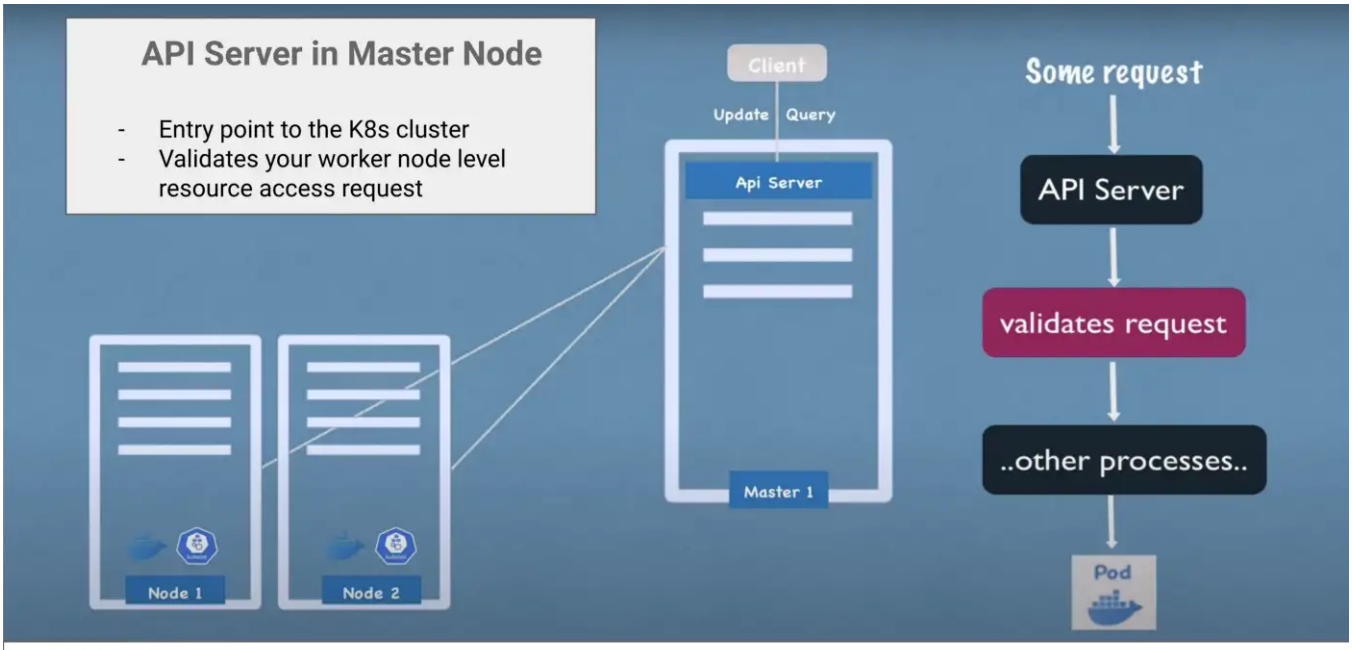
Every master nodes in the K8s cluster runs the following key processes

- **kube-apiserver**
- **kubectl: kube-controller-manager**
- **kube-scheduler**
- **etcd**

Let's look into each one of the processes in detail

kube-apiserver:

It is the main gateway to access the k8s cluster and act as the main gatekeeper for client level authentication or we can say that the **kube-apiserver** is the front end for the Kubernetes control plane.



Kube-apiserver

So whenever you want to

- Deploy any new app
- Schedule any pods or

- Create any new services
- Query status or the health of your worker nodes

You need to make a request to the API server of the master node which in turn validates your requests before you get access to the processes in worker nodes.

> *kube-apiserver is designed to scale horizontally — that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances*

kube-scheduler in K8s Master Node:

Every time as a K8s admin/developer if you want to schedule a new pod on the worker node, you need to send the request to the master API server which in turn will make a call to the Kube-scheduler process. The scheduler here will intelligently decide on which worker node this pod should be placed.

So we can define **kube-scheduler** as:

> *The key Control plane component that keep a watch for newly created Pods with no assigned worker node, and selects a worker node for them to be scheduled and run on.*

This decision to which worker node the newly created pod should be accommodated in based on the resource level availability of each node. Scheduler does the resource level query and makes critical scheduling decisions.

*The real execution of the scheduler level decision is done by the **kubelet** process in the given worker node*

Key deciding factors regarding pod schedule include:

- Individual and collective resource requirements
- Hardware/Software/Policy constraints
- Node affinity and anti-affinity specifications,
- Data locality, inter-workload interference, and deadlines.

we will understand the above constraints and policies later on as we progress to cover **k8s** in more detail.

kube-controller-manager(Kubectl):

It is one of the critical processes in a master node that monitors the status of any worker node level failures. It keeps a close watch over the event like

- Crashing of any pods in the worker node

and, requests the scheduler to restart or reschedule any dead /failed pods, after detecting such event.

These control manager component of master control planer has following types of controllers:

- **Node controller**: Responsible to respond when any worker node goes down
- **Replication controller**: It ensures that the request to maintain the correct replica count of any pod deployment is always taken care
- **Endpoints controller**: Populates the Endpoints object viz. Joins, Services & Pods
- **Service Account & Token controllers**: Create default accounts and API access tokens for new namespaces created in the worker node.

etcd in K8s Master Node:

etcd *in the master control plane is responsible to store every kind of cluster-level change in the form of key-value pairs.*

It can be easily seen as a brain of the k8s cluster which keeps the log of every minute details of changes occurring in the cluster.

For example, if any pod crashes in the worker node and it has to be rescheduled, the same gets stored in **etcd** as key-value pair, also the event of pod rescheduling on the node is also logged here.

So, the data related to some of the key questions like

- What kind of resources are available in the node?
- Did the cluster state change, due to any node failure?
- Is cluster health ok?

Is actually stored here to ensure our k8s cluster is aware of the same to take the wise actions accordingly

Note!

Application level data like DB is not stored in the etcd.

Kubernetes Component:

Now that we have understood the K8s architectural processes, it's time to look into some of the key components of K8s which helps you product-grade level container orchestration.

We will just list down those components here and will get into the details of each one of them in the second part of

" Part2: Key components & concepts of Kubernetes explained "

Some Of The Key K8s components Are :

- **Pods:** Smallest unit of k8s, which is an abstraction of the container application
- **Services & Ingress:** To manage external communication between nodes and internal pod level communication
- ConfigMaps: To manage the end-point URLs required by the pods/ DB's
- **Secrets**: To keep app-level passwords and secret keys securely using based64 encoding
- **Volume**: For Persistent data storage
- **Deployments**: To deploy create replicas & manage stateless apps
- **Statefulsets**: For stateful apps and databases

What's Next?

We will look into the concepts of each component listed above and will do small hands-on exercises to understand the implementation of each of those concepts.

So time to sign-off with this food for thought:

> *“ Any complex systems can be understood easily if you know how to break them into smaller modules, this art of simplifying things has been the core idea behind the complex Microservice architecture”*

Wishing you all a very productive 2021 and blissful #HappyNewYear

and a Big Thank You..... for being there in my journey

Kubernetes Dev Ops Cloud Computing Microservices Technology