Kubernetes Architecture explained

by TechWorld with Nana

Table of Contents

Basic architecture: Master and Worker Nodes 3
Worker Nodes - 3 processes 4
Master Nodes - 4 processes 7
Example cluster setup 12

Basic architecture: Master and Worker Nodes

Kubernetes is a complete framework, which is very powerful but at the same time very complex. 🤯 A lot of people get overwhelmed when they read the documentation of Kubernetes: How is the Kubernetes mechanism built? How do all the processes inside that mechanism work, that make it possible to manage and orchestrate the containers?

WHAT YOU WILL LEARN 💡

We look at **two types of Nodes** that Kubernetes operates on, Master and Worker nodes. You will learn **what is the difference between those two** and **which role each one of them has inside the cluster.**

We also go through the basic concepts of how Kubernetes does what it does and how its mechanism makes the cluster self-managed, self-healing and automated. Finally, you will understand how you as an operator of the Kubernetes cluster should end up having much less manual effort. 😎

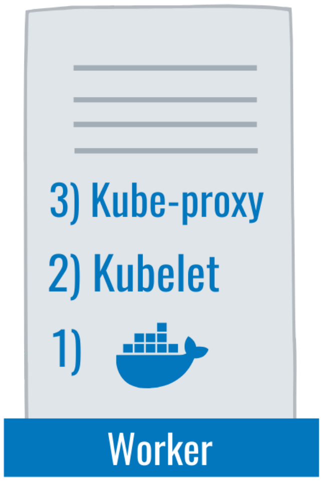
Worker
Worker
Master

Worker Nodes - 3 processes

- - - - x

One of the main components of the Kubernetes architecture is its Worker servers - which are called Nodes in Kubernetes. Each Node runs multiple application Pods with containers inside. So these pods and containers all run on worker nodes.
The way it works is that each worker node has **3 processes that must be installed on every Node:**

1) Container Runtime
2) Kubelet
3) Kube-proxy



These 3 processes are used to **schedule and manage those Pods**. Nodes are the cluster servers that actually do the work, that's why sometimes they are also called "**Worker Nodes**".

1) Container runtime 🐳

The first process that needs to run on every Node is the container runtime, for example Docker, but it could be some other technology as well.

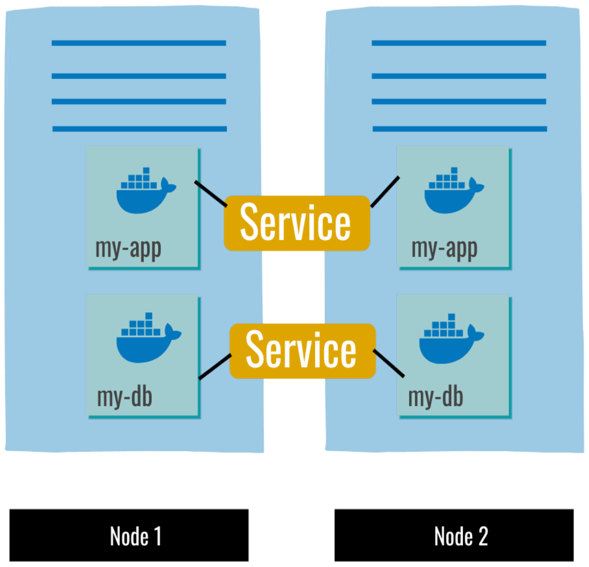
Why? 🤷‍♀️ Because application **Pods have containers running inside**, a container runtime needs to be installed on every Node.

2) Kubelet

The process that **actually starts those Pods and the containers** on the server is **Kubelet**, which is a process of Kubernetes itself, unlike container runtime. Kubelet has an **interface with both**: the container runtime and the server or the Node itself. Why? because Kubelet needs to get resources, like CPU, RAM and storage, from the worker node to create a Pod and then it needs to talk to Docker to start a container inside the pod.

Usually a Kubernetes cluster is made up of multiple Nodes, which all have container runtime and Kubelet installed. You can have hundreds of those Worker Nodes, which will run other Pods and containers or multiple replicas of the existing Pods like *my-app* and *my-db* Pods in the following example:

The way that **communication** between these pods works, is using **Services. Services** act as a **load balancer** that basically catches the requests directed to the application, like *my-db* for example and then forwards it to the one of *my-db* Pods.
And that's where the third process on the worker nodes comes into play.

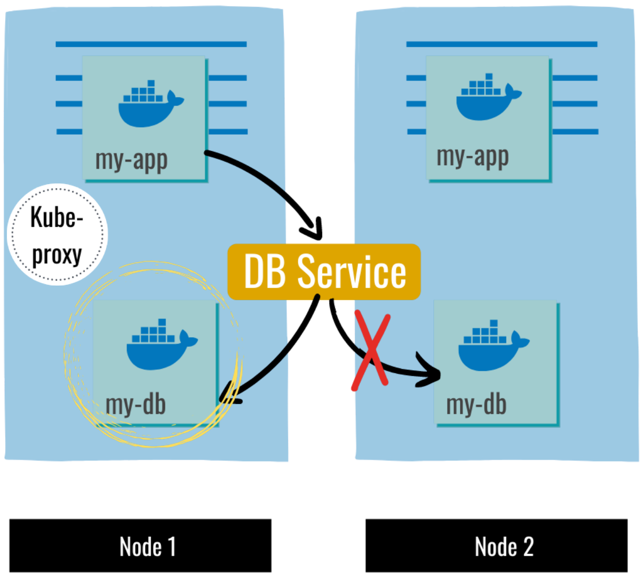


3) Kube-proxy

So, the third process that is responsible for **forwarding requests from Services to Pods** is actually Kube-proxy, that also must be installed on every Node.

Kube-proxy has actually an **intelligent forwarding logic** ⭑ that makes sure that the communication also works in the most efficient way.

For example if an application *my-app* is making a request to *my-db* application, instead of the Service just randomly forwarding the request to any replica, it will forward it to the replica that is running on the same Node as the Pod that initiated the request. This way it avoids the network overhead of sending the request to another machine.

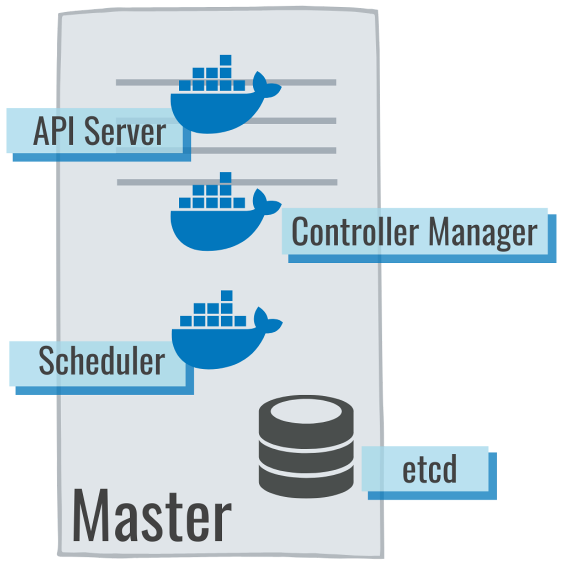


To summarize the Kubernetes processes **Kubelet** and **Kube-proxy** must be installed on every Kubernetes Worker Node along with an **independent container runtime** in order for Kubernetes cluster to function properly.

Master Nodes - 4 processes

Now the question is: ❓❓

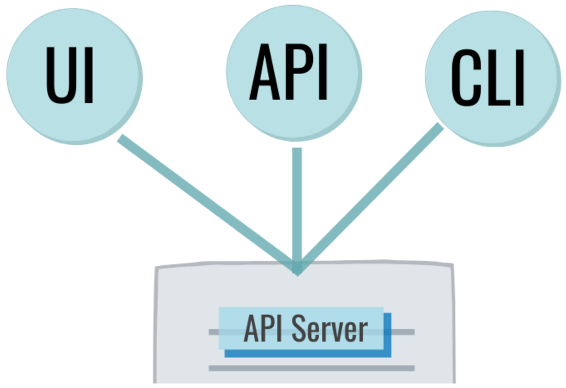
- How do you interact with a Kubernetes cluster or how do you decide on which Node a new application Pod or database Pod should be scheduled?
- Or if a replica Pod dies, what process actually monitors it and then reschedules it or starts it again?
- Or when we add another server, how does it get added to the cluster to become a new Worker Node and get Pods and containers created on it?



The answer is, all these managing tasks are done by Master Nodes. So Master servers or Master Nodes have completely different processes running on them.
There are 4 processes that run on every Master Node that **control the cluster state** and manage the Worker Nodes as well:

1) API server
2) Scheduler
3) Controller Manager
4) etcd

1) API server

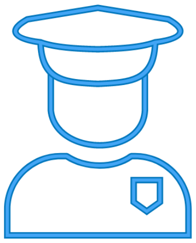


The first service is the API server. When you as a user want to deploy a new application in a Kubernetes cluster, you **interact with the API server** using a Kubernetes client.

This client can be a UI like Kubernetes dashboard or a command line tool like Kubectl.

The API server is like a **cluster gateway**, which receives the requests for creating or updating components into the cluster as well as queries about these components.

It also acts as a **gatekeeper for authentication** to make sure that only authenticated and authorized requests get through to the cluster.

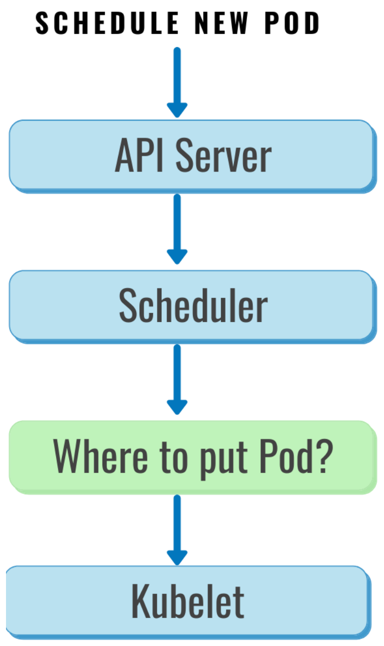


That means whenever you want to schedule a new Pod, deploy a new application, create a new service or any other component, you have to talk to the API server on the Master Node. The API server then **validates your request** ✔️ to make sure the request is valid as well as the client is authorized to make that request. If everything is fine, it will forward your request to other processes in order to schedule the Pod or create this component that you requested.
Also if you want to query the status of your Pods or the cluster health, you make a request to the API server and after the same validation, it gives you the response.

This means that with API Server, you have a **single entry point into the cluster**.

2) Scheduler

Another Master process is the Scheduler. When you send a request to an API server to schedule a new Pod, the API server - after it validates your request - will actually hand it over to the Scheduler, in order to start the application Pod on one of the Worker Nodes.



The Scheduler, instead of just randomly assigning it to any Node, has an **intelligent way of deciding** on which Worker Node the new Pod should be scheduled. 💡

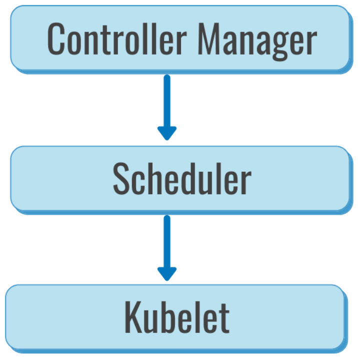
The way it works is, first the Scheduler will look at your request and see how much resources the application that you want to schedule will need, how much CPU, how much RAM etc. Then it goes through the Worker Nodes to check the available resources on each one of them.
If it sees that one Node is the least busy or has the most resources available it will schedule the new Pod on that Node.

An important point here is that Scheduler **just decides** on which Node a new Pod will be scheduled. The process that actually does the scheduling, one that actually starts that Pod with a container is the **Kubelet**. Kubelet gets the request from the Scheduler and executes the request on that Node.

3) Controller Manager

The next component is the Controller Manager, which is another crucial master process. Think about what happens when Pods die on any Node. There must be a way to detect that Pods died and then reschedule those Pods as soon as possible.

What the Controller Manager does, is it **detects state changes** like crashing of Pods for example. So when Pods die, the Controller Manager detects that and tries to recover the cluster state as soon as possible. For that it makes a **request to the Scheduler** to reschedule those Pods and the same cycle happens here where the Scheduler decides based on the resource calculation, which Worker Nodes should restart those Pods again and makes requests to the corresponding Kubelets on those Worker Nodes to actually start the Pods.



4) etcd

Finally, the last Master process is etcd, which is a **key value store of the cluster state**. You can think of it as a cluster brain, which means that every change in the cluster, for example when a new Pod gets scheduled, when a Pod dies, all of these changes get saved or updated into this key value store of etcd.

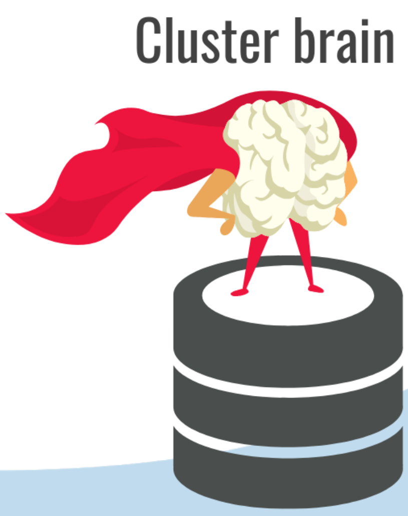
And the reason why the etcd store is the cluster brain is, because all the other processes, like Scheduler, Controller manager, Kubelet, **work** based on the **data in etcd** as well as **communicate** with each other through etcd store.
Note that Etcd can also be hosted and managed outside the cluster.

For example: ❓❓

- How does the Scheduler know what resources are available on each Worker Node?
- How does the Controller Manager know that the cluster state changed or that a pod died or that Kubelet restarted new Pods upon the request of a Scheduler?
- When you make a query to the API server about the cluster health or the state of your application, where does the API server get all this information from?

All this data is stored in the etcd store. ⭐

What is NOT stored in the etcd key value store is the actual application data. For example, if you have a database running inside the cluster, the data will be stored somewhere else. Etcd store only stores the cluster state information.



Example cluster setup

IMPORTANT!

Important Master processes

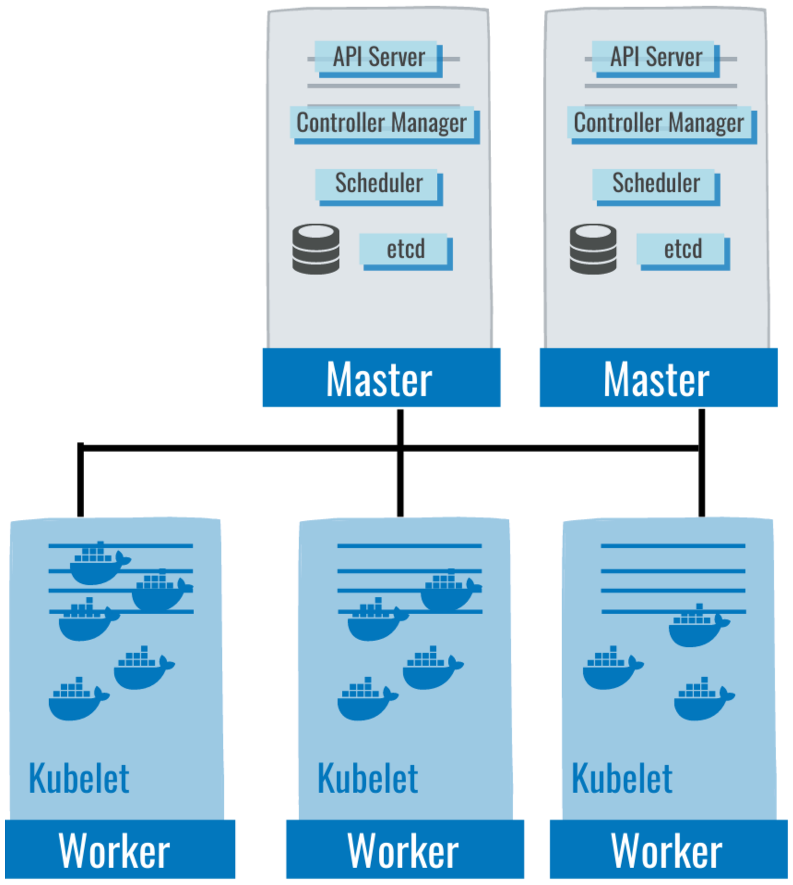
Now you probably already see that Master processes are **absolutely crucial** for the cluster operation, especially the etcd store, which contains data that must be reliably stored and replicated.

That's why in practice a Kubernetes cluster is **usually made up of multiple Masters**, where each Master Node runs its Master processes. Where the API server is load balanced and the etcd store forms a distributed storage across all the Master Nodes.

Realistic cluster setup

Now that we saw what processes run on Worker Nodes and Master Nodes let's actually have *a* look at a **realistic example** of a cluster setup.

In a very small cluster you would probably have two Master Nodes and three Worker Nodes.



Also to note here, the **hardware resources of Master and Node servers actually differ**. Master processes are more important, but they actually have less load of work so they need less resources, like CPU RAM and storage. Whereas, the Worker Nodes do the actual job of running the Pods with containers inside. Therefore they need more resources.

Increase Kubernetes cluster capacity 🚀

As your application grows and its demand for resources increases, you may actually **add more Master and Worker Nodes** to your cluster, thus forming a more powerful and robust cluster to meet your application resource requirements.

You can actually add new Master or Node nodes to the existing cluster pretty easily.

If you want to **add a Master server**:

1) you just get a fresh new server
2) you install all the Master processes on it
3) **join** it to the Kubernetes cluster using a Kubernetes command

The same way, if you need **new Worker Nodes**:

1) you get new servers
2) you install all the Worker Node processes, like container runtime, Kubelet and KubeProxy on it
3) you **join** it to the cluster

That's it. This way you can infinitely **increase the power and resources** of your Kubernetes cluster as your application complexity and it's resource demand increases. 😎

Where to go from here? 🚀

There are many great resources out there that teach you more about Kubernetes, like free videos on Youtube, or paid courses on platforms like Udemy.

On my Youtube channel [Techworld with Nana] you can also find many free videos about Kubernetes, Docker and different DevOps technologies. In addition, I also offer paid courses to go into much more detail and practical deployment scenarios.