Kubernetes Architecture

Raúl Estrada

Octubre 2020

Kubernetes

Kubernetes is the jewel of the crown of the containers orchestration. The product itself was vamped by Google leveraging years of knowledge on how to run containers in production. Initially, it was an internal system used to run Google services, but at some point, it became a public project. Nowadays, it is an open source project maintained by few companies (Red Hat, Google, and so on) and is used by thousands of companies.

At the time of writing this, the demand for Kubernetes engineers has skyrocketed up to a point that companies are willing to hire people without expertise in the field but with a good attitude to learn new technologies.

Kubernetes has become so popular due to, in my opinion, the following factors:

- It solves all the deployment problems
- It automates micro services' operations
- It provides a common language to connect ops and development with a clean interface
- Once it is setup, it is very easy to operate

Nowadays, one of the biggest problems in companies that want to shorten the delivery life cycle is the **red tape that has grown around the delivery process.** Quarter releases are not acceptable anymore in a market where a company of five skilled engineers can overtake a classic bank due to the fact that they can cut the red tape and streamline a delivery process that allows them to release multiple times a day.

This (not just this, but mainly this) is Kubernetes: a set of tools and virtual objects that will provide the engineers with a framework that can be used to streamline all the operations around our apps:

- Scale up
- Scale down
- Zero downtime rollouts
- Canary deployments
- Rollbacks
- Secret management

Kubernetes is built in a technology-agnostic way. Docker is the main container engine, but all the components were designed with interchangeability in mind: once Rkt is ready, it will be easy to switch to Rkt from Docker, which gives an interesting perspective to the users as they don't get tied to a technology in particular so that avoiding vendor locking becomes easier. This applies to the software defined network and other Kubernetes components as well.

One of the pain points is the steep learning curve for setting it up as well as for using it.

Kubernetes is very complex, and being skilled in its API and operations can take any smart engineer a few weeks, if not months, but once you are proficient in it, the amount of time that you can save completely pays off all the time spent learning it.

On the same way, setting up a cluster is not easy up to a point that companies have started selling Kubernetes as a service: they care about maintaining the cluster and you care about using it.

One of the (once again, in my opinion) most advanced providers for Kubernetes is the **Google Container Engine** (**GKE**), and it is the one that we are going to use for the examples in this book.

When I was planning the contents of this chapter, I had to make a decision between two items:

- Setting up a cluster
- Showing how to build applications around Kubernetes

I was thinking about it for a few days but then I realized something: there is a lot of information and about half a dozen methods to set up a cluster and none of them are official. Some of them are supported by the official Kubernetes GitHub repository, but there is no (at the time of writing this) official and preferred way of setting up a Kubernetes instance either on premises or in the cloud, so the method chosen to explain how to deploy the cluster might be obsolete by the time this book hits the market. The following options are the most common ways of setting up a Kubernetes cluster currently:

- **Kops**: The name stands for Kubernetes operations and it is a command-line interface for operating clusters: creating, destroying, and scaling them with a few commands.
- **Kubeadm**: Kubeadm is alpha at the moment and breaking changes can be integrated at any time into the source code. It brings the installation of Kubernetes to the execution of a simple command in every node that we want to incorporate to the cluster in the same way as we would do if it was Docker Swarm.
- **Tectonic**: Tectonic is a product from CoreOS to install Kubernetes in a number of providers (AWS, Open Stack, Azure) pretty much painlessly. It is free for clusters up to nine nodes and I would highly recommend that, at the very least, you play around it to learn about the cluster topology itself.
- **Ansible**: Kubernetes' official repository also provides a set of playbooks to install a Kubernetes cluster on any VM provider as well as on bare metal.

All of these options are very valid to set up a cluster from scratch as they automate parts of Kubernetes architecture by hiding the details and the full picture. If you really want to learn about the internals of Kubernetes, I would recommend a guide written by Kelsey Hightower called Kubernetes the hard way, which basically shows you how to set up everything around Kubernetes, from the etcd cluster needed to share information across nodes to the certificates used to communicate with `kubectl` , the remote control for Kubernetes. This guide can be found at https://github.com/kelseyhightower/kubernetes-the-hard-way.

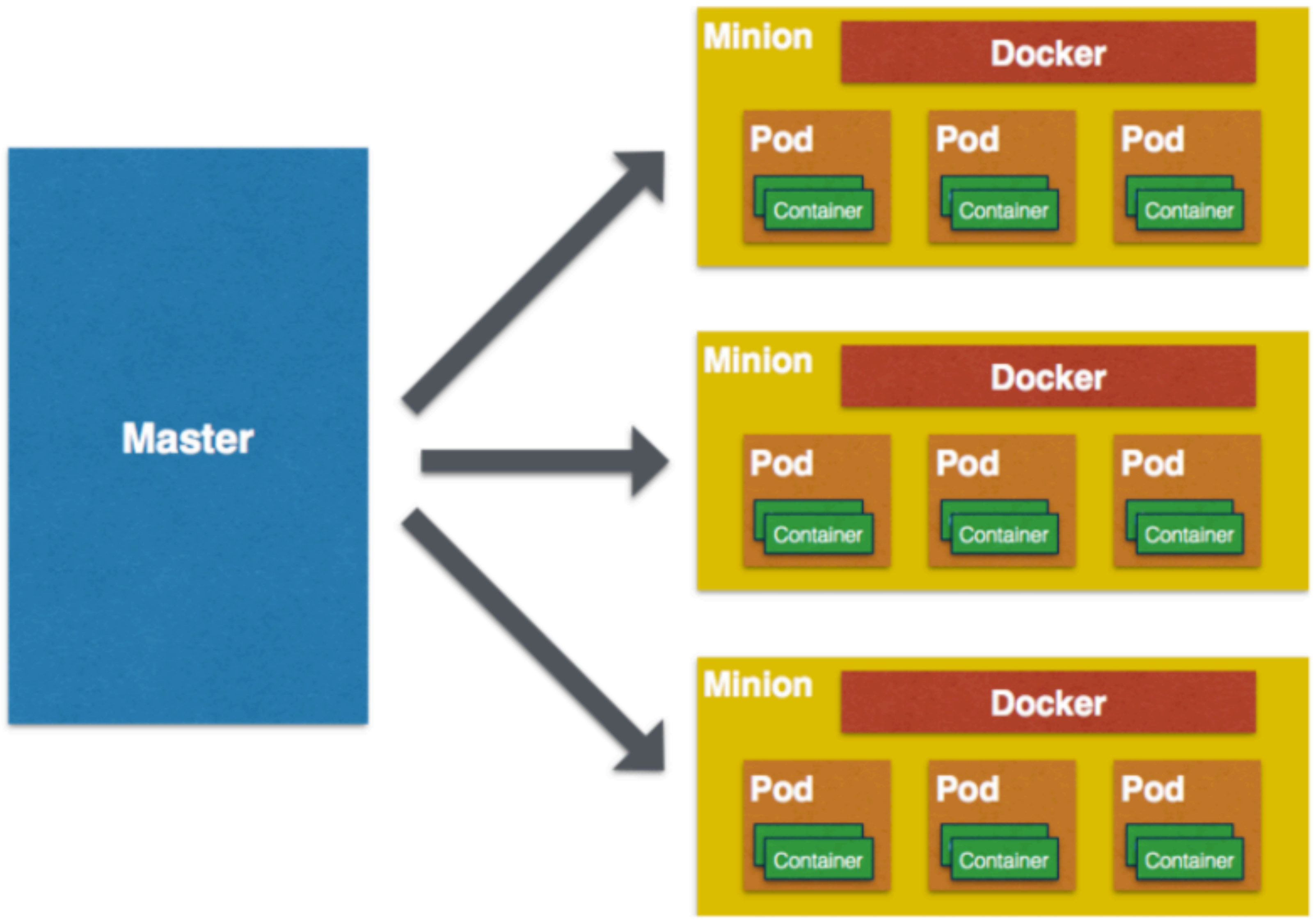
And it is maintained and up to date with new versions of Kubernetes.

As you can guess from this explanation, in this chapter, you are going to learn about the architecture of Kubernetes, but mainly, we will focus on how to deploy and operate applications on Kubernetes so that by the enc of this chapter, we have a good understanding of how we can benefit from an already running cluster.

Kubernetes logical architecture

The first problem that you will find once you start playing with Kubernetes is creating a mental map on how and where everything runs in Kubernetes as well as how everything is connected.

In this case, it took me few weeks to fully understand how it all was wiring up, but once I had the picture in my mind, I drew something similar to what is shown in the following diagram:


Master
Minion
Docker
Pod
Container
Pod
Container
Pod
Container
Minion
Docker
Pod
Container
Pod
Container
Pod
Container
Minion
Docker
Pod
Container
Pod
Container
Pod
Container

This is Kubernetes on a very high level: a master node that orchestrates the running of containers grouped in pods across different Nodes (they used to be called minions but not anymore).

This mental map helps us understand how everything is wired up and brings up a new concept: the pod. A pod is basically a set of one or more containers running in orchestration to achieve a single task. For example, think about a cache and a cache warmer: they can run in different containers but on the same pod so that the cache warmer can be packed as an individual application. We will come back to this later on.

With this picture, we are also able to identify different physical components:

- Master
- Nodes

The master is the node that runs all support services such as DNS (for service discovery) as well as the API server that allows us to operate the cluster. Ideally, your cluster should have more than one master, but in my opinion, being able to recover a master quickly is more important than having a high availability configuration. After all, if the master goes down, usually, it is possible to keep everything running until we recover the master that usually is as simple as spawning a new VM (on the cloud) with the same template as the old master was using.

The nodes are basically workers: they follow instructions from the master in order to deploy and keep applications alive as per the specified configuration. They use a software called Kubelet, which is basically the Kubernetes agent that orchestrates the communication with the master.

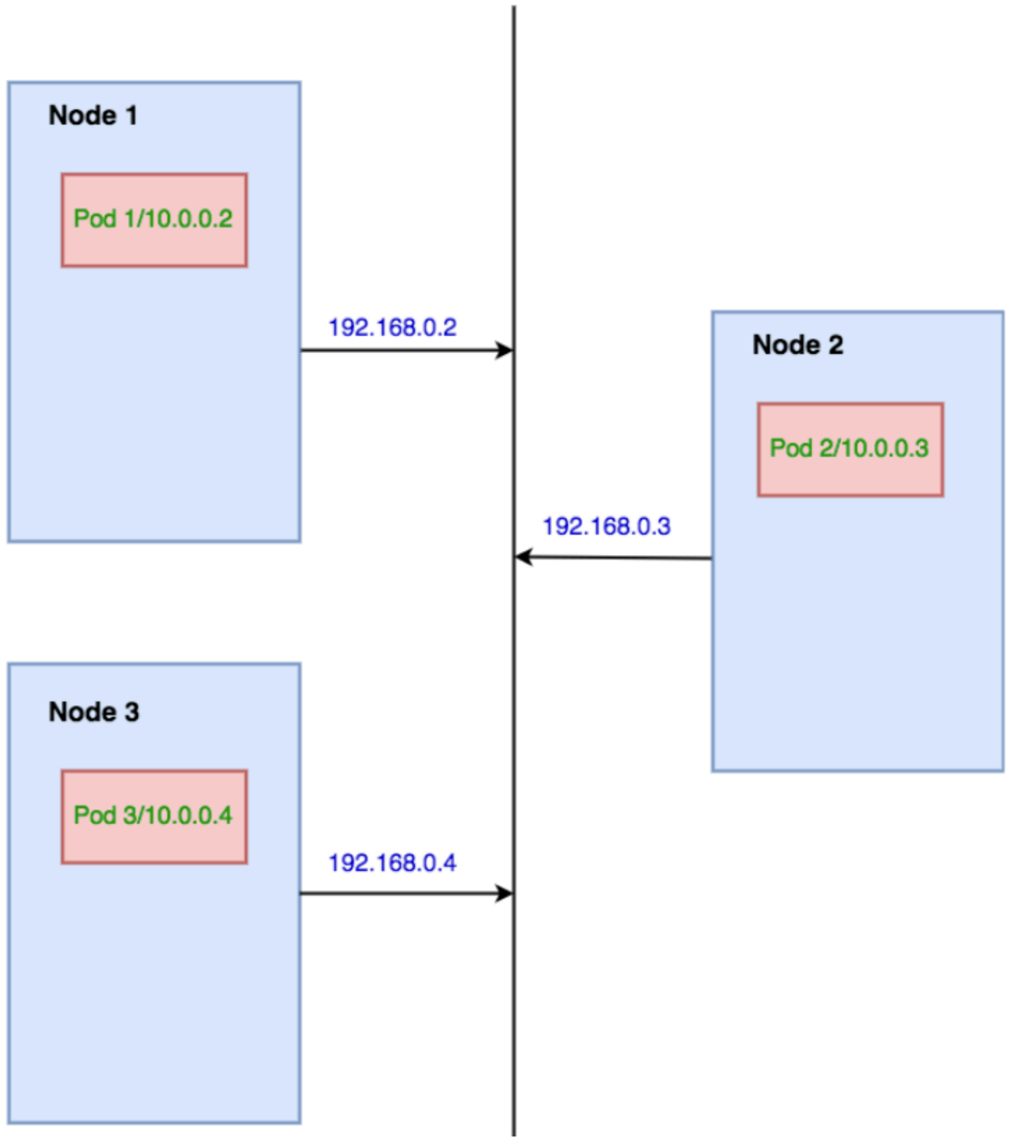
Regarding the networking, there are two layers of network in here:

- Hardware network
- Software network

The hardware network is what we all know and that is used to interconnect the VMs on the cluster. It is defined in our cloud provider (AWS, Google Cloud Platform, and so on), and there is nothing special about it, just bear in mind that ideally, this network should be a high profile network (Gigabyte Ethernet) as the inter-node traffic can be quite high.

The software network (or **Software Defined Network**, **SDN**) is a network that runs on top of Kubernetes middleware and is shared between all the nodes via **etcd**, which is basically a distributed key value storage that is used by Kubernetes as a coordination point to share information about several components.

This SDN is used to interconnect the pods: the IPs are virtual IPs that do not really exist in the external network and only the nodes (and master) know about. They are used to rout the traffic across different nodes so that if an app on the node 1 needs to reach a pod living in the **Node 3**, with this network, the application will be able to reach it using the standard `http/tcp` stack. This network would look similar to what is shown in the following figure:


Node 1
Pod 1/10.0.0.2
192.168.0.2
Node 2
Pod 2/10.0.0.3
192.168.0.3
Node 3
Pod 3/10.0.0.4
192.168.0.4

Let's explain this a bit:

- The addresses on the network 192.168.0.0/16 are the physical addresses. They are used to interconnect the VMs that compound the cluster.
- The addresses on the network 10.0.0.0/24 are the software defined network addresses. They are not reachable from outside the cluster and only the nodes are able to resolve these addresses and forward the traffic to the right target.

Networking is a fairly important topic in Kubernetes, and currently, the most common bottleneck in performance is that traffic forwarding is common across nodes (we will come back to this later on in this chapter), and this causes extra inter-node traffic that might cause a general slowdown of the applications running in Kubernetes.

In general and for now, this is all we need to know about the Kubernetes architecture. The main idea behind Kubernetes is to provide a uniform set of resources that can be used as a single computing unit with easy zero downtime operations. As of now, we really don't know how to use it, but the important thing is that we have a mental model of the big picture in a Kubernetes cluster.