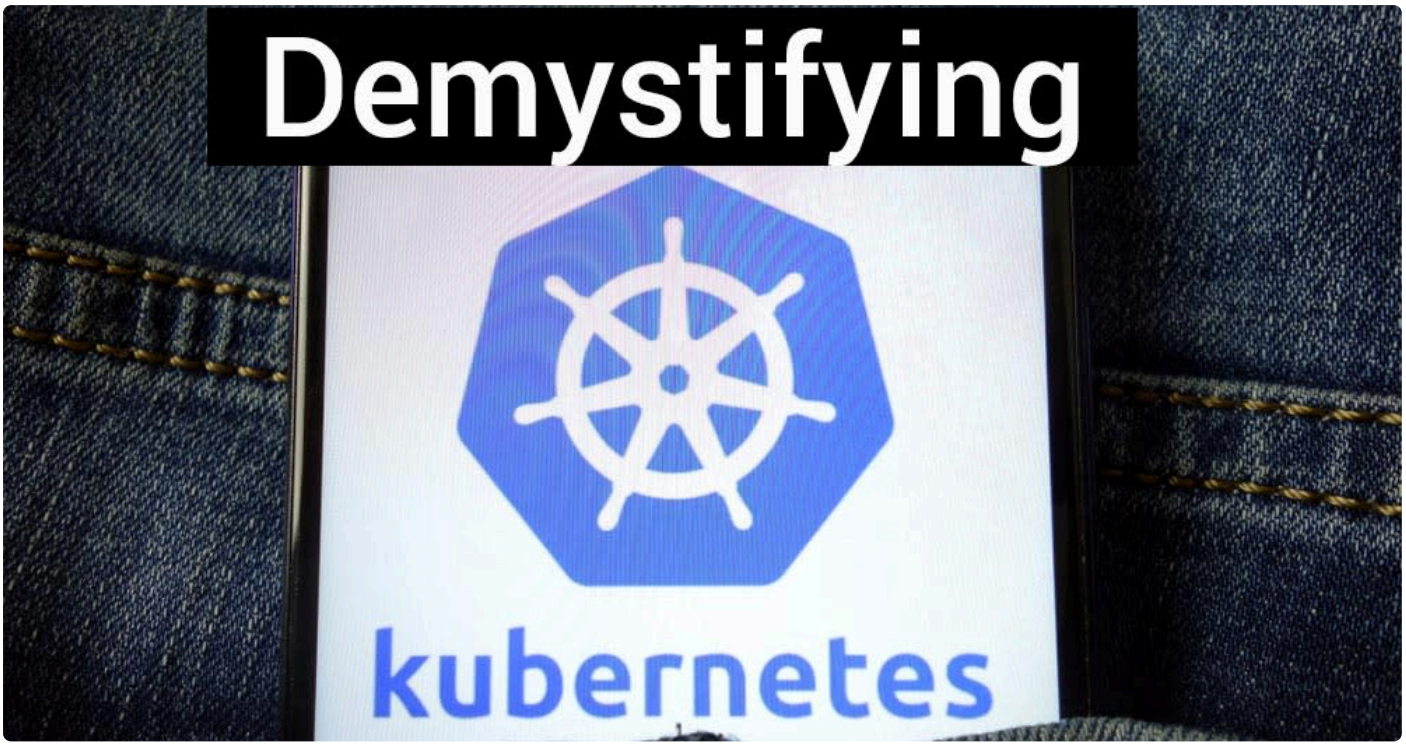
Demystifying Kubernetes

Ewa Anna Szyszka · 2020-09-17



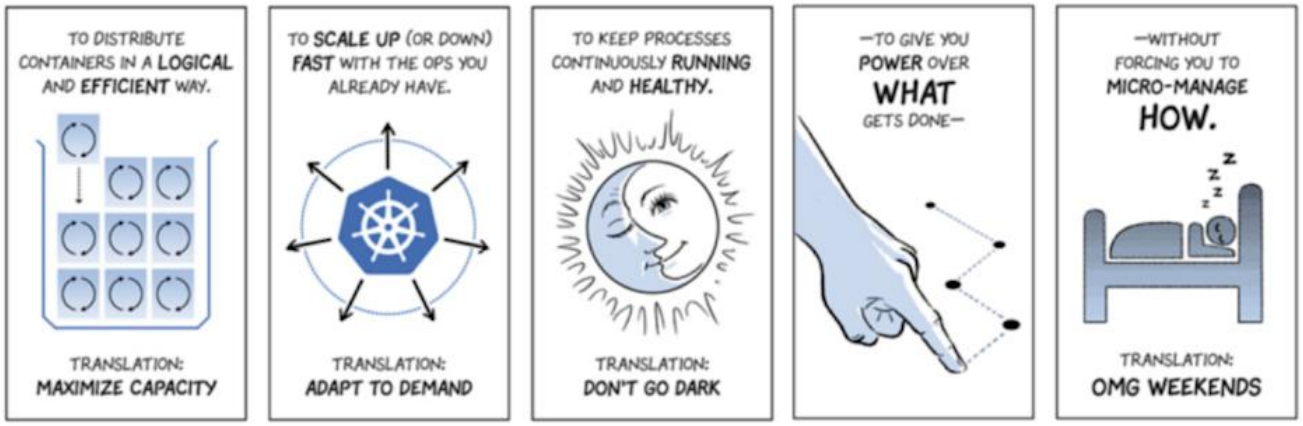
Imagine that you are working on an application that helps in emergency response. You start by developing the code on your machine. After your app is ready, you decide that you want to move your project to your **Staging environment** to test it. And finally, you deploy it to your **Production environment**.

You have a lot on your plate:Software testing, implementing new features, taking care of the security of your app, maybe even working with a team of contributors to the project from all over the world. You want to finish your application fast, and you might want to revisit your app in the future when you have more time for further improvements.

How do you ensure that the next time you work on your app you still have all the correct dependencies and the right programming language versions installed? This is where **containerization** comes into play.

Containers are a light way of wrapping together the code and its dependencies. You can think of them as packages in a warehouse ready for shipping, containing the content of the package (the code), and the address information (the dependencies).

The packages by themselves are, however, not enough. We need to have a way of orchestrating them. Or in other words, managing those packages in the warehouse. This is the role of Kubernetes [1]:



Architecture

To see how Kubernetes manages the containers (the packages in our imaginary warehouse), take a look at the graphic below illustrating the warehouse (Kubernetes).

There are two types of nodes: The **Master node** and the **Worker nod**}. The first one is responsible for the management, monitoring, and scheduling of the Kubernetes clusters. The second one hosts applications as containers.

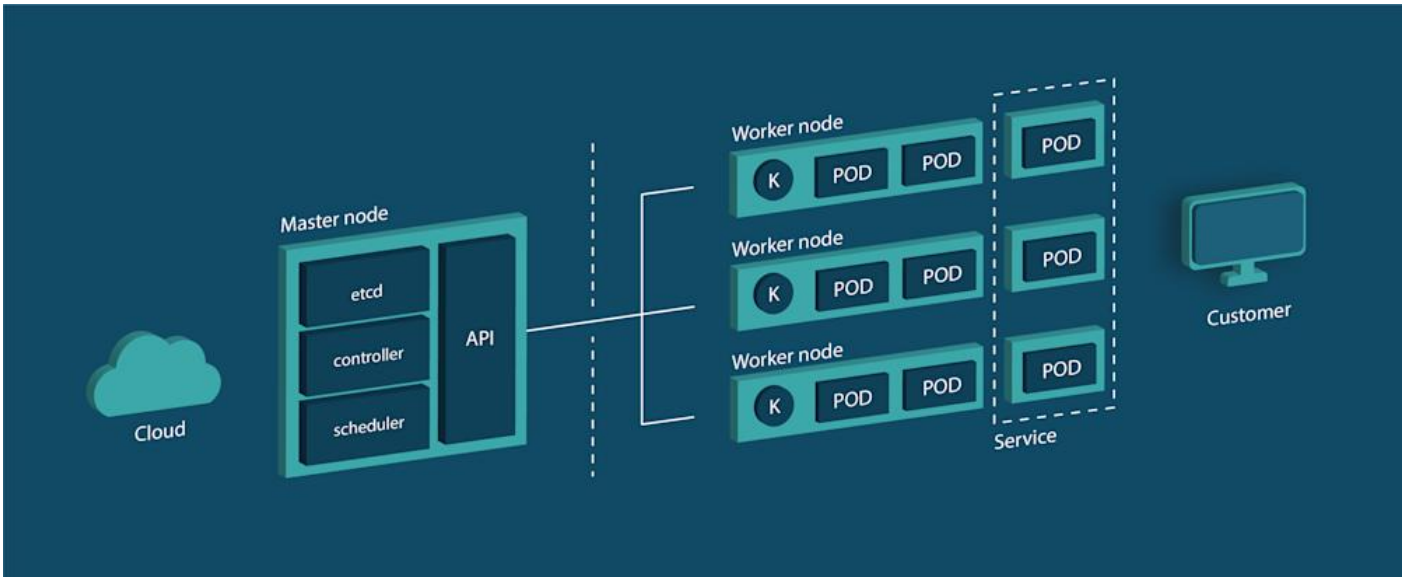
In the warehouse context, you can think of the Master node as of the control room overlooking the warehouse responsible for running various processes in the warehouse smoothly, and the Worker nodes as individual departments within the warehouse.

The Master node consists of ETCD, a database in a *key: value* format storing information about cluster states. The **Controller** ensures that the correct amount of containers is running, and it handles situations when nodes become unavailable.

It's analogous to a person that ensures that the products are in stock and in the right quantities in a warehouse. A **Scheduler** is responsible for assigning the right node to the right container based on some constraints defined. It is the crane in a warehouse that picks up packages and places them on the right shelves. Finally, we have the API, which is responsible for communication with the worker nodes.

Each Worker node consists of **Kublet**, which in logistics would be a person responsible for a particular sector of our packages. For example, the manager of priority deliveries or the manager of international shipments.

The role of the Kublet is communication and execution of instructions from the API, e.g., creating new containers (packages) or destroying some containers (cancel the package delivery). We also have PODs, which is a group of containers that are deployed together (packages shelved together).



Key architecture vocabulary

- **etcd:** distributed key-value store storing data across a cluster of servers
- **controller:** moves the current state towards the desired state
- **scheduler:** assigns a Pod to a Node
- **API:** is responsible for communication with the Worker node
- **service:** grouping of pods that are running on the cluster
- **PODs:** containers that share a common logic
- **K(ublet):** makes sure that containers are running in a pod.

Is Kubernetes right for me?

Let's look at some use cases of Kubernetes to determine if it is relevant for your project. First of all, let's ask ourselves a couple of questions:

You: Why would it be beneficial to run multiple instances of the app?

Search Guard: Running multiple instances of your app would horizontally scale it, thereby distributing the traffic and increasing your application's efficiency. Kubernetes can help you do that by managing the number of replicas of the application running, and automatically create new replicas when some of the currently running ones do not function properly using Kubernetes horizontal POD autoscaler. To give Kubernetes those instructions, you create a deployment.

You: "What is a Deployment?"

Search Guard: A Deployment is a set of instructions specified in a deployment.yaml file. The instructions describe the desired state, including the number of replicas that you want to have. In Kubernetes, you can also create multiple identical environments and distribute traffic over them (Blue-Green Deployment). You can also enable your application only for specific groups, for example, based on user characteristics (Canary Deployment), or gradually replace running instances with new ones (Rolling upgrades)

You: What are some downsides of Kubernetes?

Search Guard: Migration of your app to Kubernetes can be complicated and might not work in every case. If your application is relatively small and you are not planning on scaling horizontally it might be an overkill solution. You might want to look into alternative services and see if they will be more appropriate for your project and technical skills.

Sources

[1] Google Cloud (2019) *Smooth Sailing with Kubernetes Comic* https://cloud.google.com/kubernetes-engine/kubernetes-comic

[2] Original content of Search Guard

[3] Kubernetes.io (2020) *Official documentation. What is Kubernetes?* https://kubernetes.io/docs/concepts/

Image: shutterstock / Piotr Swat

This article was published on the Search Guard blog: https://search-guard.com/blog/demystifying-kubernetes-searchguard/