End of Studies Master's Project

Master's Thesis

Simulation of a Kubernetes Cluster with Validation in Real Conditions

Author
Théo Larue

Evaluator
Surname NAME

February 24th to August 7th 2020

Contents

Introduction / Abstract

TODO

HPC, Kubernetes and the scheduling problem

2.1 Kubernetes concepts

In the early stages of application development, organizations used to run their services on physical servers. With this direct approach came a few problematics: resources allocation, maintainability, scalability for exemple. Developers then went on with virtualized machines to run their services regardless of physical infrasctucture, which then led to the concept of containers.

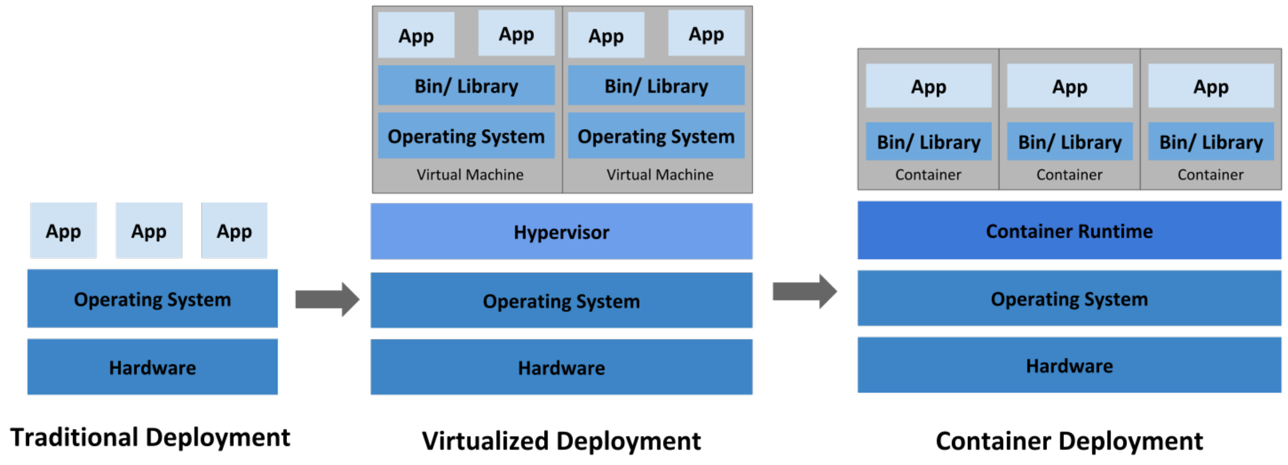

Figure 2.1: Evolution of application deployment.
Source: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

Containers can be thought of as lightweight virtual machines. Unlike virtual machines, containers share the same kernel with the host machine but still allow for a very controlled environment to run applications. There are many benefits to this : separating the development from deployment, portability, easy resource allocation, breaking large services into smaller micro-services or support of continuous integration tools (containers greatly facilitate integration tests), to cite a few.

Kubernetes[1] aims at automating of the process of deploying, maintaining and scaling containerized applications. It is an open source platform originally designed by Google and now maintained

[1]`https://kubernetes.io/`

and developed by the Cloud Native Computing Foundation[2]. It is industry grade and widely used by web services, big or small, for its - relative - ease of use and its reliability.

The basic processing unit of Kubernetes is called a **pod**. A pod is composed of one or several containerized applications and volumes (A volume is some storage space on the host machine that can be linked to containers, so they can read persistent information or store data in the long term). In the cloud native context a pod can be thought of as a service or micro-service, but we will see them a little bit differently.

Pods are bundled together in **nodes** (figure 2.2), which are most of the time physical machines (but might as well be virtual machines). They represent another layer to be passed through to access the outside world which can be useful to add some layers of security or facilitate communication between pods, for example. Each node runs at least one pod and also one kubelet which is a process responsible for communicating with the rest of Kubernetes (or more precisely, with the master node which in turns communicates with the api server). A set of nodes is called a **cluster**. Each Kubernetes instance is responsible for running one cluster.

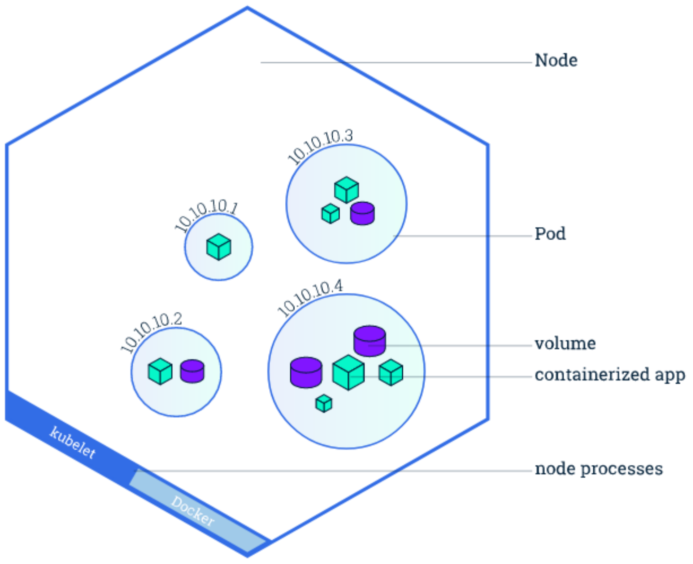

Figure 2.2: Node overview
Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

Kubernetes revolves around a central component, its api server (figure 2.3). The majority of every operations go through this api server : user interactions through kubectl, scheduling operations, saves of kubernetes state in etcd. . . We will focus on the lower-right part of this figure, that is to say the scheduler, the api server and the cluster itself.

2.2 HPC and Kubernetes

A general definition of HPC would be : "*High Performance Computing (HPC) most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer in order to solve large problems in science, engineering, or business.*" [3]

[2]`https://www.cncf.io/`
[3]`https://wwwen.uni.lu/university/high_performance_computing`

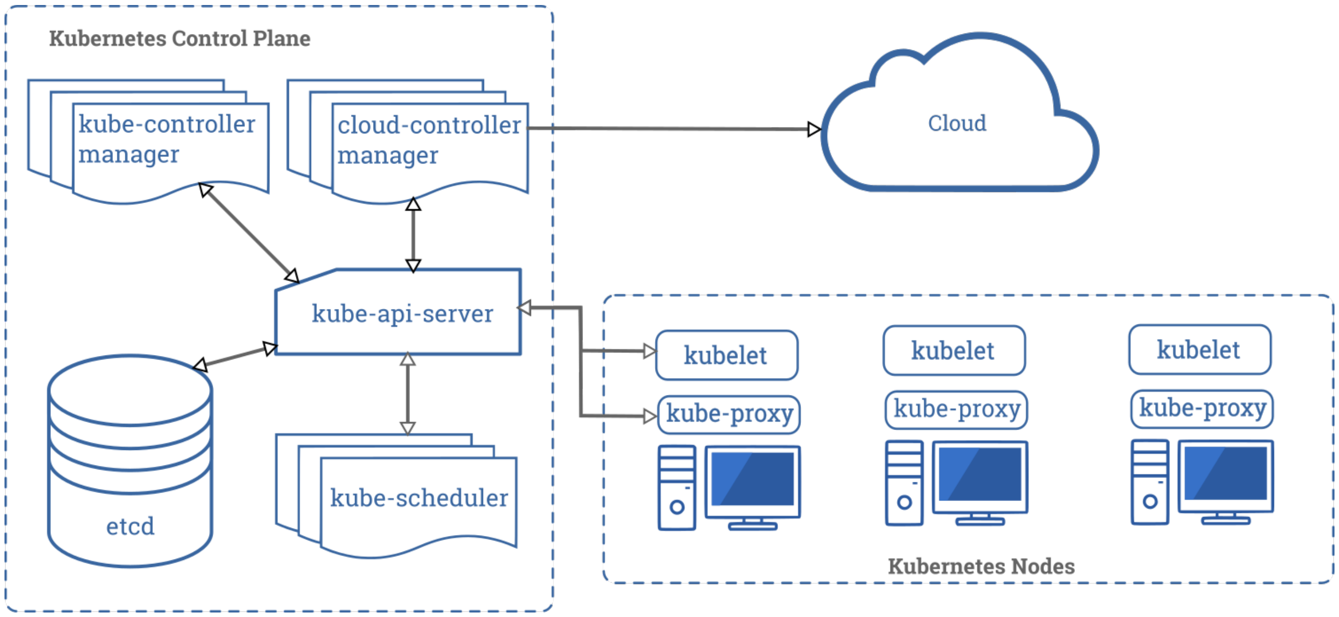

Figure 2.3: Components of Kubernetes
Source: *https://kubernetes.io/docs/concepts/overview/components/*

HPC can either refer to "High Performance Computing" or "High Performance Computer", but it is generally clear which one it refers to, given the context. An HPC workload is mostly caracterized by parallel tasks that run on a large amount of compute units - refered to as "nodes".

TODO

2.3 The scheduling problem

2.4 Simulating infrastructures

2.4.1 HPC simulators

2.4.2 Kubernetes simulation

This raises the question of scheduler development. Developing a scheduler implies being able to test its performances throughout the development process, however, testing in real conditions is time consuming and expensive. Organizations can either have enough resources to cover these costs, or test their scheduler against a simulation.

Kubernetes cluster simulations is an open problem and is the subject of this master project. Our approach relies on the Batsim[1] infrastructure simulator, which is itself built upon Simgrid[3]. Batsim is currently mostly used to simulate HPC infrastructures but was designed to be able to simulate any kind of infrastructure and therefore is theoretically able to simulate any Kubernetes cluster, moreover, Kubernetes was designed to run services but is capable of handling High-Performance Computing[2]. This project aims at adapting Batsim so it can evaluate Kuberenetes schedulers.

2.5 The Batsim infrasctucture simulator

2.5.1 Batsim concepts

2.5.2 Limitations

Problematic

3.1 Objectives

The goal of this project is to design and implement Batkube, which will be an interface between Batsim and Kubernetes schedulers. With this interface, we want to compare Batsim results gainst data from a real Kubernetes cluster, given HPC workloads.

3.2 Translation

3.3 Synchronization

Implementation

4.1 Batkube architecture

TODO

4.2 API implementation

4.3 Time hijack

TODO

4.3.1 batsky-go

Algorithm 1: Requester loop

```
   Input: req: request channel, res: result channel map
1  while Batkube is not ready do
2      wait
3  requests = []request
4  while req is not empty do
5      m = <- req /* Non blocking receive                                         */
6      requests = append(requests, m)
7  sendToBatkube(requests) /* Only requests with duration > 0 are actually sent.
       Batkube will always anwser.                                                */
8  now = responseFromBatkube()
9  for m in range requests do
10     res[m.id] <-now /* The caller continues execution upon reception           */
```

Algorithm 2: Time request (time.now())

```
   Result: Current simulation time
   Input: d: timer duration, req: request channel, res: response channel map
   Output: now : simulation time
1  if requester loop is not running then
2      go runRequesterLoop() /* There can on ly be one loop runing at a time      */
3  id = newUUID()
4  m = newRequestMessage(d, id) /* Requests are identified using uuids            */
5  resChannel = newChannel()
6  res[id] = resChannel /* A channel is associated with each request              */
7  req <- m /* The code blocks here until request is handled                      */
8  now = <-resChannel /* The code blocks here until response is sent by the
       requester loop                                                             */
9  return now
```

Evaluation

Conclusion

Bibliography

[1] *Batsim docs.* URL: `https://batsim.readthedocs.io/en/latest/`.

[2] *Kubernetes Meets High-Performance Computing.* URL: `https://kubernetes.io/blog/2017/08/kubernetes-meets-high-performance/` (visited on 04/21/2020).

[3] *Simgrid official website.* URL: `https://simgrid.frama.io/`.