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

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.

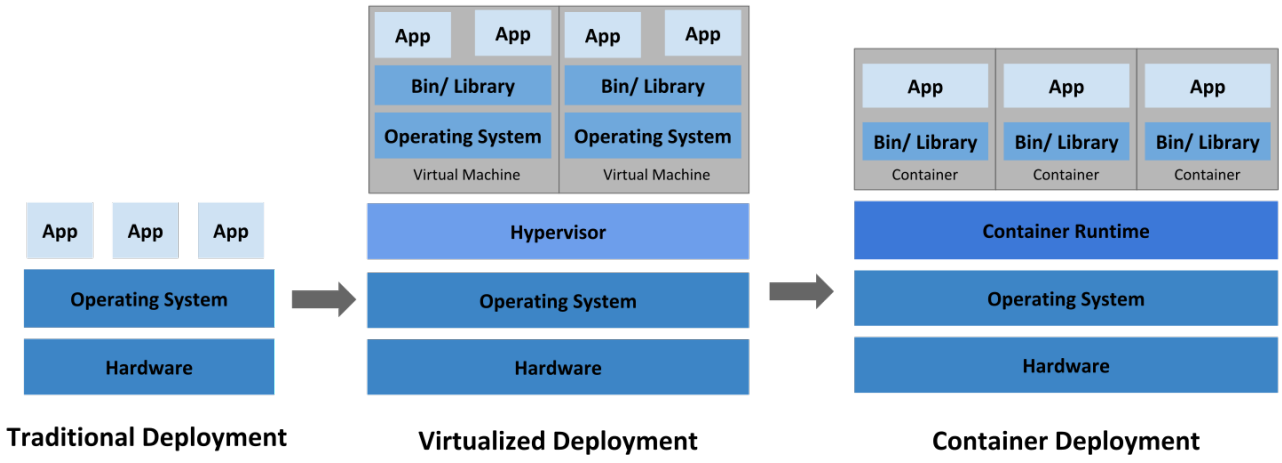

Figure 2.1: Evolution of application deployment.
Source: *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*

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.

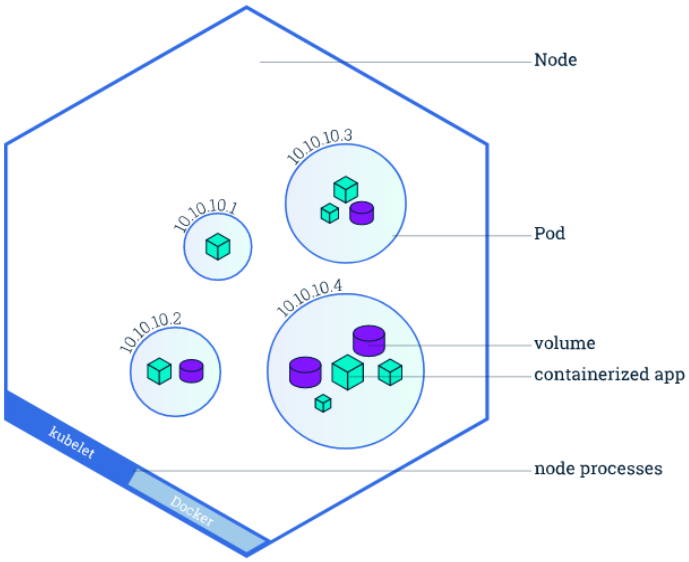

Figure 2.2: Node overview
Source: *https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/*

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`

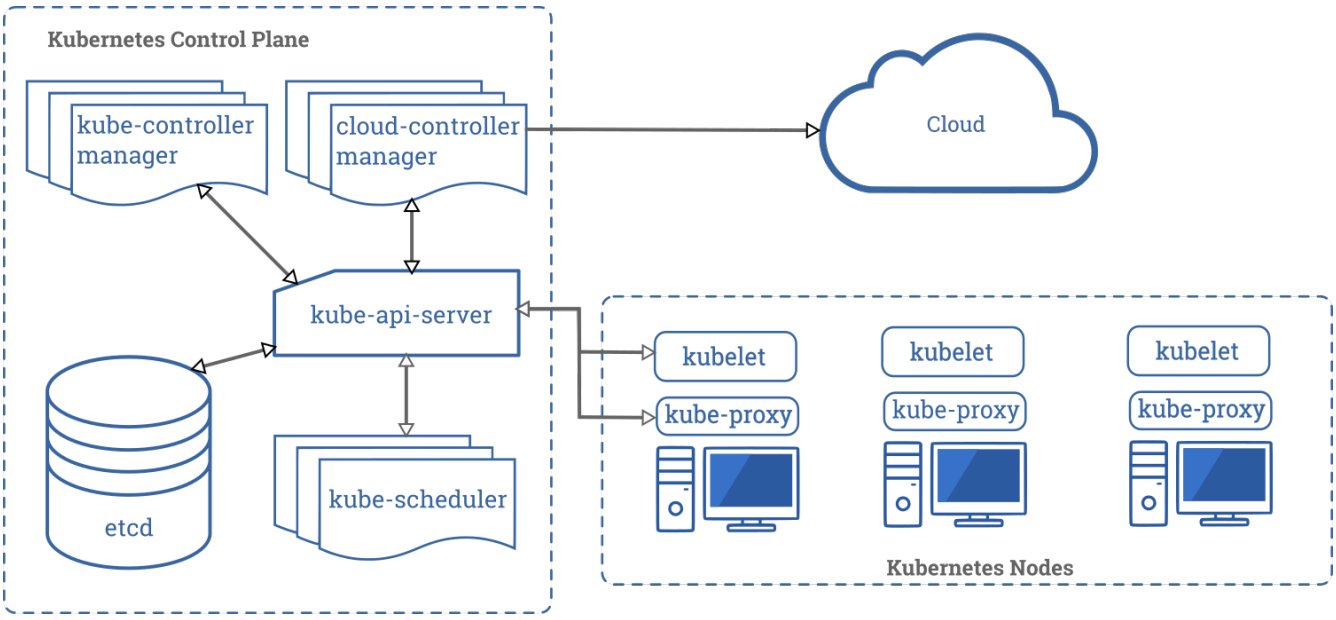

Figure 2.3: Components of Kubernetes
Source: *https://kubernetes.io/docs/concepts/overview/components/*

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".

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[3] infrastructure simulator, which is itself built upon Simgrid[10]. 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[7]. This project aims at adapting Batsim so it can evaluate Kuberenetes schedulers.

2.5 Batsim concepts

2.5.1 Limitations

State of the art

3.1 Simulating Kubernetes

Simulating the Kubernetes ecosystem is a difficult task that hasn't been done many times. In fact, we have been able to find only two references on the subject. One is a from an internship, and the other does not have any open source release.

To our knowledge, there isn't any other project about adapting Simgrid to Kubernetes.

3.1.1 k8s-cluster-simulator

k8s-cluster-simulator[4] is an internship project written in Go. It aims at simulating a Kubernetes clusters in order to evaluate schedulers with basic workloads, with as little changes to schedulers implementation as possible.

The user can specify resource usage for some pods in a yaml config file (cpu, memory, gpu) and submit them through an interface. The scheduler is also submitted through an interface, and that makes the code not exactly plug and play : the scheduler has to comply with the interface to be evaluated.

This is a simple project useful for understanding some Kubernetes core concepts and technical details.

3.1.2 JoySim simulator from JD.com

This is a project led by the company JD.com, presented in the **KubeCon + CloudNativeCon North America 2019**[1]. It looks like a very complete, fully fledged Kubernetes cluster simulator, however it is not (yet) open-source : "Planning to work with CNCF SIG-Scheduling for an open source release".

3.2 Kubernetes schedulers

Gathering knowledge on the scheduling ecosystem in Kubernetes is a first step in deciding what direction to take with Batkube. It came out that the most useful resources we could find were the very simple schedulers that were written as examples or tutorials about writing custom schedulers. These basic schedulers made the process of writing proof of concepts much easier.

3.2.1 Industry grade schedulers

Kubernetes scheduler

kube-scheduler is the de facto scheduler for any Kubernetes cluster (at least with the native Kubernetes distribution) and can be found in Kubernetes repository[8].

It is a generic scheduler and very effective in most cases.

kube-batch

kube-batch[5] is a batch scheduler developed by the Kubernetes Scheduling SIG[9].

It has proven reliable on an industrial size and serves as a base for other scheduling projects (e.g. Volcano[11]).

Poseidon

3.2.2 Example schedulers

These schedulers helped a great deal in understanding how a scheduler interacts with the Kubernetes api-server, thanks to their simplicity.

Bash scheduler

The bashScheduler[2] is a tiny project created demonstrating a very basic implementation of a random scheduler using only bash, so as to break down exchanges between the api server and the scheduler with simple http requests.

It has served as a base upon which the first POC was created.

Random scheduler

This project is taken from a tutorial[12] redacted by Banzai Cloud to guide us through the implementation of a custom scheduler using the go client[6] from kubernetes. Like the bashScheduler, it randomly binds pods on available nodes. Although, it takes things to the next level by using the go client thus introducing the concept of kubernetes informers and using https instead of plain http.

The second POC was built using this scheduler as a base.

Problematic

4.1 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.

4.2 Translation

4.3 Synchronization

Implementation

5.1 Batkube architecture

TODO

5.2 API implementation

5.3 Time hijack

TODO

5.3.1 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] *A Toolkit for Simulating Kubernetes Scheduling at Scale - Yuan Chen, JD.com.* URL: https://kccncna19.sched.com/event/UaVk/a-toolkit-for-simulating-kubernetes-scheduling-at-scale-yuan-chen-jdcom (visited on 04/21/2020).

[2] *bashScheduler repository on Github.* URL: https://github.com/rothgar/bashScheduler.

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

[4] *k8s-cluster-simulator: A simulator for evaluating Kubernetes schedulers.* URL: https://tech.preferred.jp/en/blog/k8s-cluster-simulator-release/ (visited on 04/21/2020).

[5] *kube-batch repository on Github.* URL: https://github.com/kubernetes-sigs/kube-batch.

[6] *Kubernetes Go Client repository on Github.* URL: https://github.com/kubernetes/client-go.

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

[8] *Kubernetes repository on Github.* URL: https://github.com/kubernetes/kubernetes.

[9] *Kubernetes Scheduling SIG.* URL: https://github.com/kubernetes/community/blob/master/sig-scheduling/README.md.

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

[11] *Volcano repository on Github.* URL: https://github.com/volcano-sh/volcano.

[12] *Writing custom Kubernetes schedulers.* URL: https://banzaicloud.com/blog/k8s-custom-scheduler/ (visited on 04/21/2020).