

Master of Science in Informatics at Grenoble
Master Informatique
Specialization MoSIG

Simulation of a Kubernetes Cluster with Validation in Real Conditions

LARUE Théo

Defense Date, 2020

Research project performed at Laboratoire d'Informatique de Grenoble

Under the supervision of:

Michael Mercier

Defended before a jury composed of:

Head of the jury

Jury member 1

Jury member 2

Abstract

TODO : remove the focus on HPC

The rise of containerized applications has provided web platforms with much more control over their resources than they had before with their physical servers. Soon enough, developers realized they could go even further by automating container management operations to allow for even more scalability. The Cloud Native Computing Foundation was founded in this context, and developed Kubernetes which is a piece of software capable of container orchestration, or in other words, container management. Now, as we observe a convergence between HPC (High Performance Computing) and the Big Data field where Kubernetes is already the standard for some applications such as Machine Learning, discussions about leveraging containers for HPC applications rose and interest in Kubernetes has grown in the HPC community. One of the many challenges the HPC world has to face is scheduling, which is the act of allocating tasks submitted by users on available resources. In order to properly evaluate and develop schedulers researchers have used simulators for decades to avoid running experiments in real conditions, which is costly both in time and resources. However, such simulators do not exist for Kubernetes or are not open to the public. While the default scheduler works great for most of the Cloud Native infrastructures Kubernetes was designed for, some teams of researchers would rather be able to experiment with different batch processing policies on Kubernetes as they do with traditional HPC. Our goal in this master thesis is to describe how we developed Batkube, which it is an interface between Kubernetes schedulers and Batsim, a general purpose infrastructure simulator based on the Simgrid framework and developed at the LIG.

Acknowledgement

I would like to express my sincere gratitude to .. for his invaluable assistance and comments in reviewing this report... Good luck :)

Résumé

Abstract mais en français

Contents

Abstract	i
Acknowledgement	i
Résumé	i

1	Introduction	1
1.1	The increasing demand in computing infrastructures	1
1.2	Studying infrastructures through simulation	1
1.3	Contribution	1
2	State of the art	3
2.1	Infrastructure simulators	3
2.2	Kubernetes schedulers	3
3	Implementation	5
3.1	Batsim	5
3.2	Kubernetes	5
3.3	Objectives	8
3.4	Technical challenges	9
3.4.1	Translation	9
3.4.2	Time hijack	9
3.4.3	Re-building the API	9
4	Evaluation	11
5	Conclusion	13

Introduction

1.1 The increasing demand in computing infrastructures

Computation is everywhere in our interconnected world. The online service industry is now omnipresent (one could even say, omniscient) in our daily lives , and GAFAs now require massive amounts of computational power and storage. We cannot know for sure how much data it represents now - we can always try to guess ¹ - as these companies do not disclose statistics on their size, however a study[1] has shown global electricity usage for the information and communications technology field could rise to a fifth of the global electricity usage by 2025 despite the great breakthroughs in building energy efficient computing. This alone shows how much the demand in infrascture (both in storage and computing power) is increasing.

TODO : demand in industry, research, services, public domain, cryptocurrencies types of infra : hpc, hpda, gpu farms, distributed systems

1.2 Studying infrastructures through simulation

TODO virgin real life experiments vs chad simulated scenarios

1.3 Contribution

Batkube, and interface between Batsim infrastructure simulator and Kubernetes

¹<https://what-if.xkcd.com/63/>

State of the art

2.1 Infrastructure simulators

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[**batsim**] infrastructure simulator, which is itself built upon Simgrid[**simgrid**]. 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[3]. This project aims at adapting Batsim so it can evaluate Kubernetes schedulers.

2.2 Kubernetes schedulers

kube-scheduler

Not exactly a batch scheduler, it is the default scheduler for Kubernetes made by the CNCF"" -> "hop là"

kube-batch

Poseidon

bashScheduler by rothgar

random-scheduler by Banzaicloud

k8s-custom-scheduler by IBM

scheduler by kelseyhightower

Implementation

3.1 Batsim

3.2 Kubernetes

Kubernetes overview

In the early stages of application development, organizations used to run their services on physical servers. With this direct approach came many challenges that needed to be coped with manually like resources allocation, maintainability or scalability. In an attempt to automate this process developers started using virtual machines which enabled them to run their services regardless of physical infrastructure while having a better control over resources allocation. This led to the concept of containers which takes the idea of encapsulated applications further.

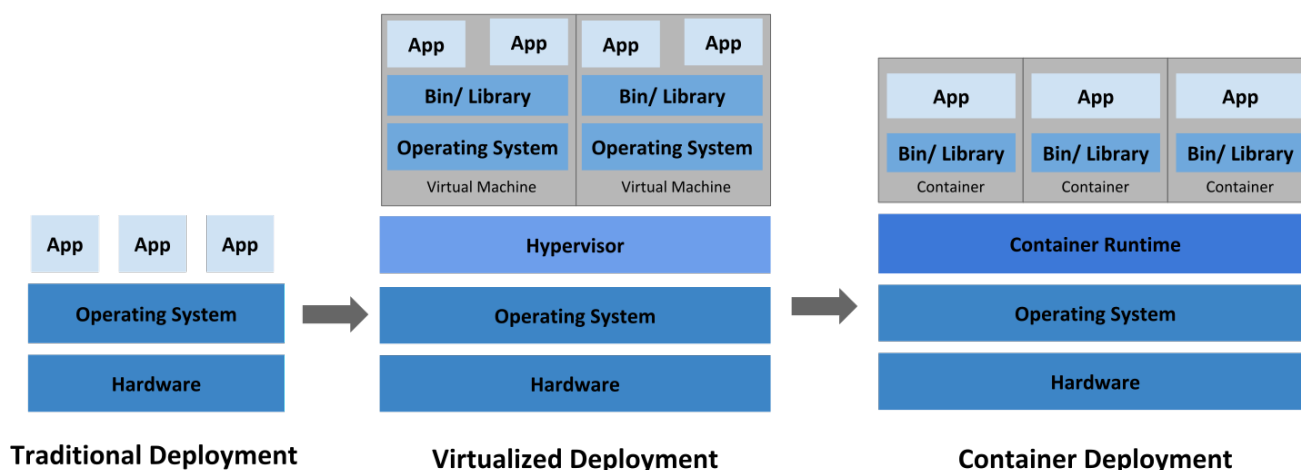


Figure 3.1: Evolution of application deployment.

Source: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Containers can be thought of as lightweight virtual machines. Unlike the latter, 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).

The CNCF¹ (Cloud Native Computing Foundation) was founded in the intent of leveraging the container technology for an overall better web. In a general way, we now speak of these containerized and modular applications as cloud native computing :

“Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”²

Kubernetes³ is the implementation of this general idea and was announced at the same time as the CNCF. It aims at automating of the process of deploying, maintaining and scaling containerized applications. It is industry grade and is now the de-facto solution for container orchestration.

The basic processing unit of Kubernetes is called a **pod** which is composed of one or several containers and volumes⁴. In the cloud native context a pod most often hosts a service or micro-service.

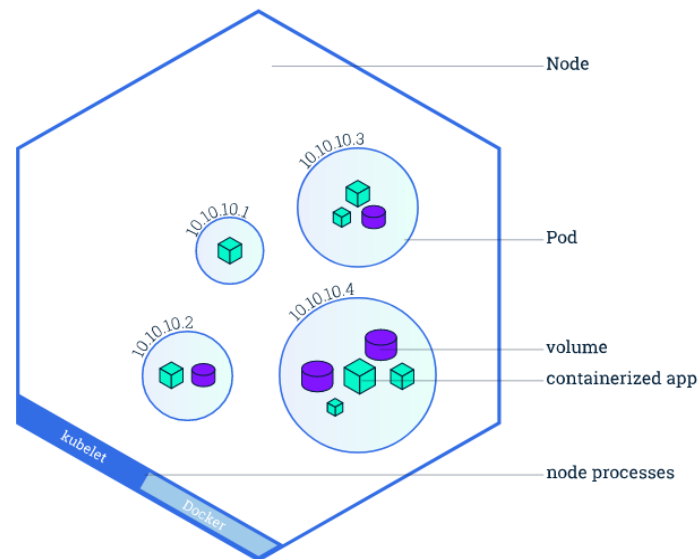


Figure 3.2: Node overview

Source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

Pods are bundled together in **nodes** (figure 3.2) which are either physical or virtual machines. They represent another barrier to pass through to access the outside world which can be useful to add layers of security or facilitate communication between pods. Nodes take the idea of containerisation further by encapsulating the already encapsulated services. 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 a cluster.

Kubernetes revolves its API server which is its central component (figure 3.3). The majority of operations between components go through this REST API like user interactions through kubectl or scheduling operations.

¹<https://www.cncf.io/>

²<https://github.com/cncf/toc/blob/master/DEFINITION.md>

³<https://kubernetes.io/>

⁴A 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

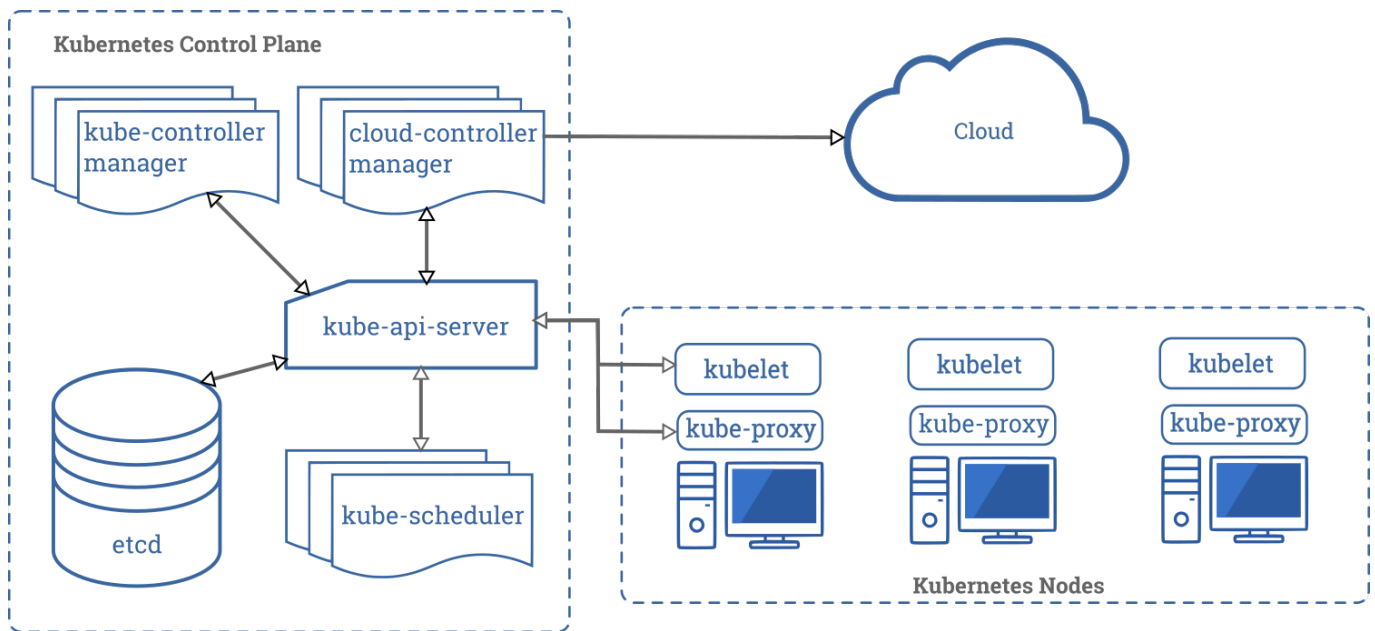


Figure 3.3: Components of Kubernetes

Source: <https://kubernetes.io/docs/concepts/overview/components/>

HPC and Kubernetes

The difference between HPC and Cloud Native computing lies in the workloads they are intended to tackle. Kubernetes was designed for Cloud Native applications. Services or micro services are run in containers and are expected to be available at all times : they are replicated as many times as the user desires and restarted whenever a failure occurs. High availability is at the core of Kubernetes container management. On the other hand, depending on scheduling policies, HPC is focused on user wait time, maximizing resource usage, optimizing energy costs... For instance, in case of failure, it is sometimes not sufficient to restart the single job that failed : the entire submission must be re-run if it is part of several jobs computed in parallel.

Kubernetes is now the standard for AI and Machine Learning as shown by the many efforts at making this coupling an efficient environment[4][6][5], which brought an increasing interest for container driven HPC aswell and Kubernetes for HPC in particular. Batch schedulers such as kube-batch⁵ have been implemented for kube, and numerous HPC applications like slurm⁶ now support containers as well.

Indeed, containers have many advantages that HPC users can benefit from. Here are some notable ones:

- First off, research has shown that Kubernetes offer similar performance to more standard bare metal HPC[2].
- Users will get the same environment everywhere making up for a uniform and standardized work-place.
- Portability : users could seamlessly hop from one infrastructure to another based on their needs and criteria like price, performance, and capabilities rather than compatibility.
- Encapsulation : HPC applications often rely on complex dependencies that can be easily concealed into containers.

Despite all those advantages, Kubernetes is not ready yet to be used in proper HPC environment because it lacks vital components like a proper batch job queuing system, and support for MPI applications. It cannot

⁵<https://github.com/kubernetes-sigs/kube-batch>

⁶<https://slurm.schedmd.com/containers.html>

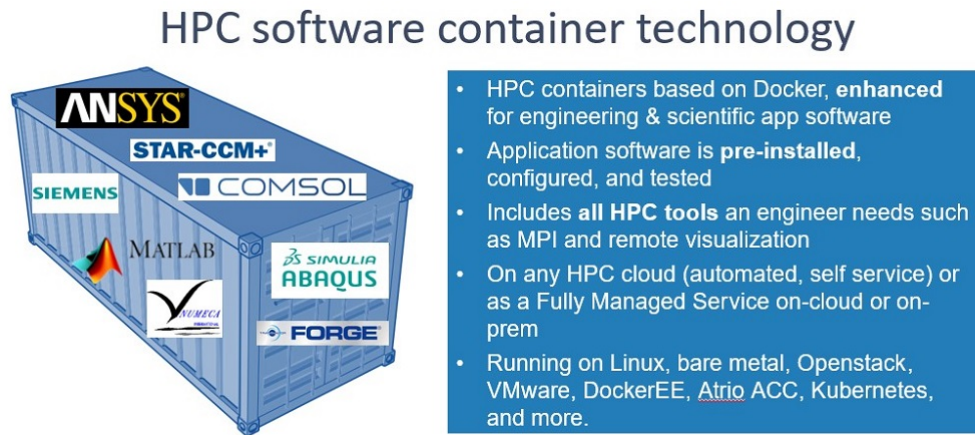


Figure 3.4: The container technology for HPC

Source: <https://www.hpcwire.com/2019/09/19/kubernetes-containers-and-hpc/>

yet compete against the very well established HPC ecosystem, but that time may come soon as containers are becoming more and more integrated in modern infrastructures.

3.3 Objectives

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

3.4 Technical challenges

3.4.1 Translation

3.4.2 Time hijack

TODO

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 answer. */
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

```

3.4.3 Re-building the API

— 4 —

Evaluation

Conclusion

Bibliography

- [1] Anders Andrae. “Total consumer power consumption forecast”. In: *Nordic Digital Business Summit* 10 (2017).
- [2] A. M. Beltre et al. “Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms”. In: *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. 2019, pp. 11–20.
- [3] *Kubernetes Meets High-Performance Computing*. URL: <https://kubernetes.io/blog/2017/08/kubernetes-meets-high-performance/> (visited on 04/21/2020).
- [4] Mikyoung Lee, Sungho Shin, and Sa-Kwang Song. “Design on distributed deep learning platform with big data”. In: (2017).
- [5] Seetharami R. Seelam and Yubo Li. “Orchestrating Deep Learning Workloads on Distributed Infrastructure”. In: *Proceedings of the 1st Workshop on Distributed Infrastructures for Deep Learning. DIDL ’17*. Las Vegas, Nevada: Association for Computing Machinery, 2017, 9–10. ISBN: 9781450351690. DOI: 10.1145/3154842.3154845. URL: <https://doi.org/10.1145/3154842.3154845>.
- [6] Boris Tvaroska. “Deep Learning Lifecycle Management with Kubernetes, REST, and Python”. In: Santa Clara, CA: USENIX Association, May 2019.