UCLouvain

École polytechnique de Louvain

Improving the performance and the scalability of INGINIOUS

Author: **Guillaume Everarts de Velp**
Supervisor: **Ramin Sadre**
Readers: **Olivier Bonaventure, Anthony Gégo**
Academic year 2019–2020
Master [120] in Computer Science

Acknowledgements

Guillaume Everarts de Velp, 2020

Abstract

Contents

Chapter 1

Introduction

In this chapter I will simply introduce the subject of this master thesis, showing some basic concepts related to it and some key aspects.

1.1 INGInious

INGInious is a web platform developped by the UCL. It is a tool for automatic correction of programs written by students. It currently relies on Docker containers to provide a good isolation between the machine hosting the site and the execution of the student's codes. So that a problem in a program submitted by a student couldn't have any impact on the platform. Docker also allow to manage the resources granted to each code execution. For now INGInious can meet the demand and provide honest performances and responsiveness. But looking at the growing usage of the platform, we might soon come to a point where we reach the limits of the current implementation.

1.1.1 Architecture

INGInious counts four main components:

1. The front-end: the website with which each student interacts when submitting a task.

2. The back-end: a queue of all the tasks that need to be graded.

3. The docker agent: responsible for the container assignment to the pending tasks when resources are available.

4. The docker containers: one for the student code and one for the teacher tests evaluating the student's code behaviour.

The journey of a task submitted on INGInious is represented on Figure 1.1.

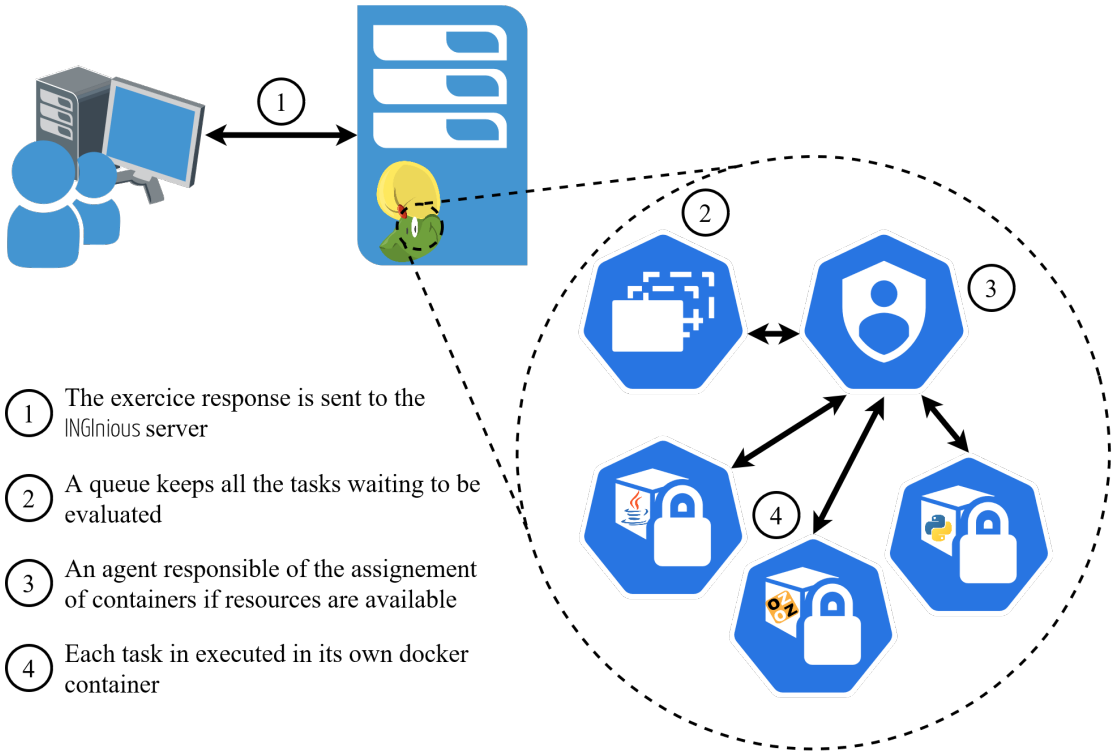

Figure 1.1: INGInious global architecture

1.1.2 Key features

The key features of INGInious, that allow it to meet the requirement of a code grading plateform are the following:

- Isolation between the student's code and the platform.
- Resource limitation (CPU, RAM, Network) for the student's code execution.
- Modularity and versatility regarding the tasks that INGInious can correct. Multiple programming language are supported, and new ones can easily be added.

1.1.3 Bottlenecks

When a task is submitted, we can count five delays before the answer can be delivered to the student:

- The sending time: the time it takes for the task to be sent to the back-end.
- The waiting time: the time the task will spend in the queue, waiting for an available container.
- The booting time: the time it takes to the container to boot and be ready to evaluate the task.
- The grading time: the time it takes for the code to run and for the teacher's container to grade it.
- The response time: the time to send the reponse back to the student.

For the first and the last one, supposing that the machine hosting the website is not overwhelmed, the delay depends entirely on the network, this is a bit out of our hands here. The waiting time is directly related to the current load on the platform, this is a more a symptom of the server overwhelming than its cause, it could be directly solved by using a scaling strategies (see section 1.2). And then we come to the booting time and the grading time, which directly depends on the containerization technology used and on the hardware performances, this is were we are going to try to improve things in this thesis.

1.2 Scaling

Currently, the resources provided to INGInious vary depending on the load that the platform is expected to be facing. Typically, when the grading of an exam is done by INGInious, the platform is scaled up, and during the holidays it is scaled down. When it comes to scaling, two strategies can be used; vertical scaling and horizontal scaling.

Vertical scaling consists in adding more resources on a single machine, to improve its performances when needed. For example, when the number of tasks arriving to the server grow, we could increase the number of virtual Cores allocated to the Virtual Machine hosting the platform in order to be able to threat more of them concurrently. If the size of the waiting queue is increasing, we might want to make more RAM available.

Horizontal scaling consists in sharing the workload across multiple machines, so that each machine can handle a small part of it. This is a solution widely used nowadays as it allows to scale up virtually indefinetely, which is not the case with vertical scally where we depend on the maximum capacity of the hardware. This requires to rethink the architecture of the platform globally.

1.3 Intentions

The master thesis aims at improving INGInious, regarding its performances and its scalability. To do so, I will search and compare different containerization technologies that could be used instead of Docker. The goal is to find an alternative that decreases the booting time (and the grading time) without loosing any of the key features of the platform. If such an alternative is found and proven to be worth the change, INGInious could then be refreshed with it.

Chapter 2

State of the art

In this chapter I will present how things are today in the field of containerization. I will start with a quick reminder of the difference between virtualization and containerization, then go through some existing solutions for both of those concepts.

2.1 Virtualization vs Containerization

Both virtualization and containerization aims at providing an abstraction layer that allows the application or the OS (Operating System) located above the abstraction to behave like if it was the only one present on this layer. The key difference between those two is where this abstraction layer is located.

To keep it simple, for the virtualization, the abstraction is between the OS and the hardware, the OS thinks it is running on a baremetal machine but is actually hosted on another machine. For the containerization, the abstraction is between the appliciation and the OS, the application thinks it is the only one running in the system but it is actually only isolated in its own namespace. As a good illustration is better than a thousand words, Figure 2.1 represents this difference.

Because of what they are, those two solutions usually didn't apply to the same situations. A VM (Virtual Machine) is a much more complicated and complete structure, that takes more time to boot but provides a better isolation. And the Container is much lighter, quicker to launch but less isolated. Traditionally, in the Cloud Computing world, VMs rely in IaaS (Infrastructure-as-a-Service) while Containers rely in PaaS (Platform-as-a-Service). This paper [4] made a detailed point about it back in 2014.

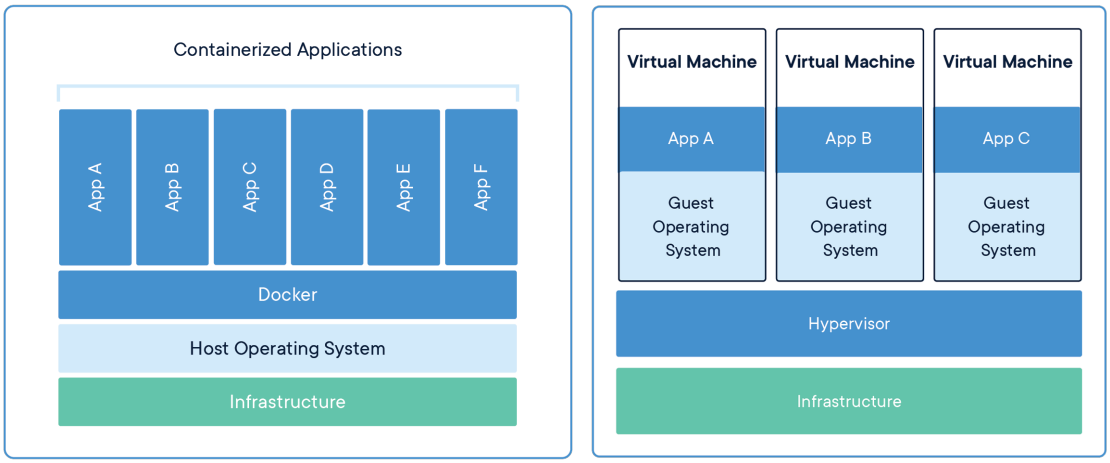

Figure 2.1: This image comes from Docker's website [3].

But recently things started to change, some new solutions seem to have the advantage of both sides, the isolation of a VM and the portability and lightness of a Container. This is why some VM-based solution are presented here. Even though VMs do not come from the same domain of application as Containers, and even though in the case of INGInious, Containers were 5 years ago the obvious choice, it might now be time to reconsider it.

2.2 Namespaces

Namespaces are a feature of the Linux kernel since 2002, they are a key element that made containerization possible. A namespace can be associated to a context, which is a partition of all the resources of a system that a set of processes has access to, while other processes can't. Those sets of resources can be of seven kinds:

- **mnt** (Mount): This controls the mount points. Processes can only have access to the mount point of their namespace.
- **pid** (Process ID): This allows each process in each namespace to get a process id assigned indepedently of other namespaces porcesses.
- **net** (Network): This provides a virtualized network stack.
- **ipc** (Interprocess Communication): This allows processes of a same namespace to communicate with one another, for example by sharing some memory.

- **uts** (Hostname): This allows to have different hostnames on a same machine, each hostname being considered as unique by the processes of its namespace.
- **user** (User ID): This allows to pretend to change the user in a namespace, si that it could behave like he had more privileges for example.
- **cgroup** (Control group): This provides a virtualization of the resources available to the processes of the namespace.

When a Linux machine starts, it initiate one namespace of each type in which all the processes run. The processes can then choose to create and switch of namespaces.

2.3 Containerization solutions

Here bellow are presented some containerization solutions.

2.3.1 Docker

Docker [6] is today probably the most known solution for containerization. Since its apparition in 2014, its interest for the PaaS sector hasn't ceased to grow. It is now tightly binded with Kubernetes, "an open-source system for automating deployment, scaling, and management of containerized applications" [2]. It consists in a daemon, running on the host, that allows to easily manage different containers. It relies on Containerd, "An industry-standard container runtime with an emphasis on simplicity, robustness and portability" [1], which itself uses runC (we will talk about it later).

The main advantages of Docker are its simplicity of use, its production-grade quality, the huge fleet of ready-to-run containers publicly available and a lot of useful tools (like `docker-compose` that come along with it). This is the solution currently used by INGInious.

2.3.2 Podman

2.3.3 runC

2.3.4 SAND

2.3.5 SOCK

[7]

2.4 Virtualization solutions

Here bellow are presented some lightweight virtualization solutions.

2.4.1 Firecracker

2.4.2 Kata Containers

2.4.3 LightVM

[5]

2.5 Summary

Chapter 3

Measurements

Chapter 4

Results

Chapter 5

Conclusion

Bibliography

[1] An industry-standard container runtime with an emphasis on simplicity, robustness and portability. `https://containerd.io/`. Accessed: 2020-02-26.

[2] Production-grade container orchestration. `https://kubernetes.io/`. Accessed: 2020-02-26.

[3] What is a container? `https://www.docker.com/resources/what-container`. Accessed: 2020-02-24.

[4] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. Virtualization vs containerization to support paas. In *2014 IEEE International Conference on Cloud Engineering*, pages 610–614. IEEE, 2014.

[5] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My vm is lighter (and safer) than your container. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 218–233, 2017.

[6] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[7] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. {SOCK}: Rapid task provisioning with serverless-optimized containers. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 57–70, 2018.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**