POLITECNICO

MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

WIP Title: Dynamic resource allocation in the cloud for compute heavy tasks in a containerized environment

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING

Author: **Elia Ravella**

Student ID: 967243
Advisor: Prof. Raffaela Mirandola
Co-advisors: Name Surname, Name Surname
Academic Year: 2021-22

Abstract

Keywords: Cloud, Containers, Dynamic infrastructure

Contents

1 | The Problem, the State of the Art and Current Available Solutions

1.1. Introduction

1.2. Containerized Environment and High Performance Computing

Containerization Virtualization is "the act of creating a virtual (rather than actual) version of something at the same abstraction level, including virtual computer hardware platforms, storage devices, and computer network resources"[1]. In cloud environments virtualization is the most used tool to provide isolated services, as compute capabilities, storage capabilities or networking. There are different kinds of virtualization, depending on the layer virtualized: some solutions just virtualize the hardware and let the user install a full fledged OS over it, other virtualizes all the technological stack from the hardware to the OS level, leaving to the user (in most of the cases, a developer) only the problem of developing the application s/he wants to ship, without the need to care about hardware limitations or operative systems settings.

One of the most used virtualization technique in the industry is containerization: when a software is said to be *containerized* it is packaged in a format that encapsulates also all its dependencies, configuration files and variables and OS settings. This package, the container, must be run through a container engine which provide the communication with the underlying software and OS to execute the software. Famous container engines are Docker, Podman, chroot or rkt. Containerized applications have several benefits over classic deployed ones:

- They do not rely on the machine or OS they are running, just the engine

- If the container has been built correctly, the configuration of the application is already done

- Packaging all their dependencies makes them independent from the other application installed on the system: two containerized applications running on the same engine could be using the same library but at different (and even incompatible) versions

- The container acts also as an isolation mechanism, that keeps the application from interacting with other systems: this improves the security of the container applications

If an application is containerized, moreover, it is very easy to deploy several instances of the same application working at the same time: this kind of deployments improve scalability and reliability of the application.

High Performance Computing   With "high performance computing" is usually addressed the field in computer science that studies computational heavy problems and develops solutions to solve them via techniques as extreme parallelism, clusterization and high performance networking. In the early stages of this project I focused also on HPC in order to understand if an virtualization approach as the containers one is suitable for such kind of tasks. The problem was the additional layer of virtualization added by the container engine: since most of the times HPC software relies heavily on low-level procedure calls, the additional virtualized layer could degrade too much the performance. However, as stated in the paper "Exploring the support for high performance applications in the container runtime environment"[2] (which original aim was to compare native performance with containerized ones) the optimization of engines is reaching a level that can offer near-native performance for HPC-like loads, at least in not-extreme scenarios.

1.3.   State of the Art

1.3.1.   Shifter

1.3.2.   SLURM

1.3.3.   Kubernetes

1.3.4.   Serverless Approach

1.4.   The Problem

This section highlights the problems of the currently available solutions: the focusing on scaling through replication rather than on resources size, and the problem of having a dynamical *in two senses*, both resource- and replication-wise, computational layer

2 | Design and Testing Phase

2.1. MapNCloud Original Architecture

Here I talk about the original deployment of the MapNCloud service. I plan to add a subsection explaining in detail the tech stack.

2.2. Problems Addressed

1. database choice and API modification

2. queue monitoring

3. resizable backend containers

4. cloud provider integration

At the end of this section I will present the "final" design draft

2.3. Testing and Validation

HERE I will introduce the "diffusion analysis" to justify the test parameters

1. CouchDB testing

2. RabbitMQ testing

3. Cloud providers options, pros and cons

4. technological limitations (docker-compose, load balancers)

I will also present the real "final" Architecture that will be deployed here, with cloup provider's technological names and services

3 | Implementation

3.1. Frontend

3.2. Backend

3.3. Database

3.4. Messaging Middleware

3.5. Computational Layer

3.5.1. Renderino

Bibliography

[1] Virtualization — Wikipedia, the free encyclopedia. URL `https://en.wikipedia.org/w/index.php?title=Plagiarism&oldid=5139350`. [Online; accessed 28-July-2022].

[2] J. P. Martin, A. Kandasamy, and K. Chandasekaran. Exploring the support for high performance applications in the container runtime environment. 2017.