Universidade da Coruña

Universidade de Santiago de Compostela

MASTER IN HIGH PERFORMANCE COMPUTING

MASTER'S THESIS

Viability analysis of infrastructure managers for the automated deployment of virtual HPC clusters on Cloud environments

Mariadalit Rosales Rodríguez

Director
Xoán Carlos Pardo Martínez

Santiago de Compostela, July 26, 2022

Acknowledgments

En el borde de mi camino no hay una silla, hay cientos, esperando a que me siente, pero avanzo con confianza y una sonrisa porque me siento ligera al caminar. Noto la mano de la gente que me quiere en mi espalda empujándome hacia la meta, una ayuda con la que no todo el mundo cuenta.

Para muchos este camino puede significar un mero paseo por un prado verde y llano, pero no para mí. Este ha sido un camino de retos, con bosques ofreciéndome sombra, con charcos que saltar y algún que otro asfalto por el que caminar. En cualquier caso, ha sido emocionante y enriquecedor.

Gracias a todos y cada uno de los profesores que he tenido durante este Máster. Vuestro tiempo y dedicación consiguen que gente corriente como yo demos pasitos hacia delante.

Gracias a mi tutor, Xoán, por darme esta oportunidad, por regalarme más tiempo del que podría pedirle y por sus consejos.

Gracias a mi padre por hablar mi mismo idioma, por ayudarme incondicionalmente y aguantarme.

Gracias a mi madre por todo el cariño y tiempo que me regala cada día para que me mantenga optimista y feliz.

Gracias a mis hermanas y a mi abuela por ser mis compañeras de viaje y conseguir que nunca me sienta sola.

Gracias a mi amor, Javier, por apoyarme de manera inagotable, por estar ahí en todos los momentos, por tu cariño y, sobre todo, por motivarme para ser una mejor persona.

Gracias a todos por darme vuestro tiempo y amor. En definitiva, eso es lo que realmente ha hecho posible que llegue aquí.

Abstract

Due to the increase of hybrid environments and the rise of Cloud resources in the IT sector, we have also seen an increase in the complexity of the administration we face today. Today, more than ever, automation tools are needed to facilitate not only the creation but also the management of these new environments. The lack of standardization, especially for HPC systems, forces us to present a proposal to evaluate the different tools that claim to facilitate the automation of this type of systems in Cloud environments and a subsequent analysis of them. In this Master's Thesis we will discover the importance of Cloud environments in HPC systems, the challenges that this presents and an analysis methodology that will serve as a reference for administrators and IT professionals who need to manage such complex and dynamic environments as HPC clusters in Cloud environments.

Keywords: HPC, Cloud, IaC, "Over IaC", IM, Elasticluster, Terraform, Ansible.

Resumen

Debido al aumento de entornos híbridos y al auge de los recursos Cloud en el sector IT, se ha podido observar también un incremento en la complejidad de administración de los mismos. Hoy más que nunca son necesarias las herramientas de automatización que faciliten no sólo la creación sino también la gestión de estos nuevos entornos. La falta de estandarización especialmente para los sistemas HPC fuerza a presentar una propuesta de evaluación de las distintas herramientas que presumen de facilitar la automatización de este tipo de sistemas en entornos Cloud y un posterior análisis sobre las mismas. En este Trabajo Final de Master descubriremos la importancia de los entornos Cloud en los sistemas HPC, los retos que ello presenta y una metodología de análisis que sirva de referencia para administradores y profesionales del sector TI que se vean en la necesidad de gestionar entornos tan complejos y dinámicos como los clústeres HPC en entornos Cloud.

Palabras clave: HPC, Cloud, IaC, "Over IaC", IM, Elasticluster, Terraform, Ansible.

Index

Figure Index

Table Index

1. Introduction

High Performance Computing, or HPC, is at the highest and most important point in its history. Without knowing it, the whole society depends to some extent on its proper functioning and evolution, which makes its development essential since it is related to the evolution and development of society. HPC has always focused on developing the highest performance systems, with special focus on parallel programming, parallel processing and low latency networks, together with the complex administration that this parallelism entails. These great features have allowed researchers from all over the world, and with sufficient budget, to carry out simulations, modeling or designs that would have been impossible if they had needed real creations to be carried out. In addition, HPC is always a window to the creation of new innovations and discoveries that allow society to advance, which means that there are applications of HPC systems in all existing socioeconomic sectors, as found in the well-known TOP500 site[1]. A clear example of recent times is how HPC has unequivocally helped to accelerate the fight against the COVID-19 pandemic, from the simulation of its spread by means of aerosols, to the processing of the genetic code of the coronavirus causing the disease to find the protein that needed to be inhibited by the vaccine[2].

Improving the accessibility of HPC environments to researchers and organizations is therefore a guarantee that society will grow to achieve its best version.

1.1 Motivation

This growing need for HPC systems, along with their rapid growth and change, has been accompanied by an unprecedented increase in the amount of data they must process, the size of the analytical models containing that data, and supporting development environments of various types. Where initially only C or FORTRAN development was done, we now find heterogeneous frameworks such as CUDA or OpenCL and modern high-level languages such as Python. This has meant that the accessibility of classic high-capacity HPC environments has been reduced, mainly due to the costs derived from the exorbitant initial investment required to have a modern HPC cluster[3].

It is precisely these very high initial costs that make HPC a technology that is still prohibitive for many researchers and organizations that do not have suitable HPC environments, either because of economic limitations (high initial costs, only occasional needs…) or because they have older HPC clusters and therefore lower capacities. This makes it mandatory to have accessible, flexible and capable environments that allow from creating an HPC environment from scratch, to expand the capabilities of an existing environment. And this is where the Cloud has the capacity to solve this problem.

The Cloud has accelerated this evolution because it is scalable and elastic, allowing self-service provisioning of one to thousands of processors in minutes. As a result, HPC users are going to cloud environments with new and expanding application

[1] "Application areas" option from https://www.top500.org/statistics/sublist/
[2] https://ieeecs-media.computer.org/media/marketing/conferences/sc21-report-v2.pdf
[3] https://www.researchgate.net/publication/258580735_HPC_in_Abstract

requirements and are seeing reduced time-to-results, faster speed to deployment, greater architectural flexibility, and reduced costs. Cloud Computing is pushing HPC at the pace of computing innovation as users benefit from advances in microprocessors, GPUs, networking, and storage.

Aware of this need that they are able to fill, the main cloud providers, Microsoft Azure, Amazon Web Services and Google Cloud Platform, have developed unique tools that allow their users to create and deploy HPC clusters. However, far from giving freedom to their users, they create dependencies on the cloud environment itself, making it difficult to interconnect with other external systems and also with private clouds such as OpenStack.

For this reason, there is a clear need to put on the table to analyze the options of existing tools on the market to achieve a flexible HPC environment, with the lowest possible cost. This tool should simplify cluster administration tasks, either for its creation from scratch, or to expand the capabilities of an existing cluster regardless of whether they will be permanent or ephemeral environments, with a particular focus on minimizing the learning curve of the administrators of these platforms.

As expected, the development of these hybrid environments poses a new challenge for system administrators and operators to control and prepare the flexible and customized environment that users demand. The administrators of HPC environments are not only subject to the conventional tasks of managing operating system errors, network failures or storage failures, but their daily work requires them to have knowledge of scientific software and libraries and the indispensable tools for parallel programming. To this complexity we add that in recent years several companies and organizations in the HPC market have decided to incorporate cloud resources to their on-premise infrastructure, creating hybrid environments and adding new elements to the managed infrastructure. It is known that automation largely solves this new challenge, but there is still no standard evaluation procedure and recommendations on the tools to implement to facilitate the task of administrators of such specific environments as HPC clusters.

In the task of finding tools that make it easier for administrators and users to deploy and manage virtual HPC clusters, we find two types of tools: IaC (Infrastructure as Code) tools and software that makes use of these IaC tools with the aim of facilitating the deployment of HPC clusters, which we will call "over IaC" as explained in following sections. However, difficulties and drawbacks are encountered when making use of any type of these tools. The former, despite their great capabilities, are not designed to specifically deploy virtual HPC clusters, while the latter, although they seem to greatly facilitate the deployment of these clusters, lack the flexibility to customize the environments they deploy and, being technology at a very early age, require a higher learning curve and therefore their adoption is very complex.

Taking into account the above, this Master's Thesis aims to establish a baseline analysis to study the feasibility of the different tools available on the market that have the ability to create flexible, on-demand and automatic virtual HPC clusters in cloud environments without dependencies on specific manufacturers, with multi-cloud and hybrid capabilities.

1.2 Objectives

The previous section has oulined the advantages offered by the cloud environment to the new needs of HPC users, forcing the evolution of these traditional environments. Adopting the characteristics of on-demand scalability, cost per use and deployment of complex environments in just minutes compared to the months it usually takes to assemble a traditional infrastructure, HPC systems deployed in cloud environments become of great interest not only for conventional users of this type of environments but for new researchers and professionals from multiple sectors who have the need to provide a result of their work in the shortest possible time and with unconventional system requirements.

With the above in mind, in this Master's Thesis it will be performed an analysis of the feasibility of different tools or "virtual managers" that allow users and administrators to automate the deployment of a virtual HPC cluster in cloud environments, with the aim of not only determining what tools exist in the market with the capabilities described, but also to establish a basis on which to build an analysis that will help find the ideal tool, detecting the shortcomings of this type of paradigm along the way. Since this is a complex objective, an attempt will be made to break it down into small objectives to be met.

Therefore, the essential objectives to be addressed in this final project are:

1. Identify whether it makes sense to deploy high-performance systems, i.e. HPC clusters, in private, public or hybrid cloud environments.
2. Whether there is a reliable method to help evaluate infrastructure management tools for automated deployment of HPC clusters in cloud environments.
3. If there is any leading tool in the market that stands out from the rest in terms of how feasible its implementation, adoption and development is, to manage these environments, based on the method of the first question.

To meet the essential objectives, the following issues will be addressed:

- Show the advantages of adopting the cloud computing paradigm in HPC systems.
- Describe the different types of tools available to automate the deployment and teardown processes of an HPC cluster in cloud environments.
- Perform a primary classification of the available tools in the market as a purely IaC tool or manager of virtual environments that make use of IaC in their underlying layers and which we call "over IaC" tools.
- Perform an introductory experiment that will allow us a first contact with these tools and focus the analysis on the most attractive ones for our scenario.
- Decide which metrics are the most relevant to administrators when deciding whether or not to implement one of those tools to design a reliable research method.
- Design the infrastructure characteristics of the HPC cluster inside the OpenStack cloud to serve as a reference prototype during the analysis.
- Evaluate the main HPC deployment automation tools in the predesigned virtual cluster to find blocking points, advantages and disadvantages of each of them according to the designed methodology.
- Answer the main objectives and establish a starting point for future studies on this topic.

1.3 Structure of the document

This document has attempted to follow a defined structure with the aim of making its reading as simple as possible, so that its content can be easily followed. To this end, the document has been divided into the following sections:

Chapter 1 introduces the problem prosecuted by this Thesis. After defining it, the different objectives that are being followed are defined. This chapter also includes the motivation for such a project.

Chapter 2 has attempted to provide the reader with a state-of-the-art review of the possibilities of Cloud Computing, its contribution to HPC and the difficulties associated with cluster deployment. In addition, an overview of the various IaC and "over IaC" tools that enable automated deployment in cloud environments is provided, with special emphasis on the deployment of virtual HPC clusters in cloud environments.

After having presented the theoretical framework and technologies, in *Chapter 3*, and due to the scant information found on the analysis and operation of "over IaC" tools, an experiment will be established to learn about the capabilities offered by these tools in relation to the deployment of custom HPC clusters. This will allow establishing the basis on which to build the analysis and also select the tools to be compared with greater criteria.

Chapter 4 aims to define different alternatives on how to carry out the analysis. This section, therefore, will define the tools to be analyzed and the reasons for doing so, together with the method chosen for their analysis. Additionally, the desirable characteristics of the HPC environment deployment tools will be established, classifying them and describing the different aspects that compose them. In addition, an HPC cluster to be deployed will be established in order to measure the possibilities of the tools selected for the analysis.

In *Chapter 5* the necessary development and implementation will be carried out for the deployment of the virtual HPC cluster using the different selected tools. Along with this development, the analysis of the tools will be carried out following the criteria defined in *Chapter 4* with the aim of obtaining data for the different defined metrics.

Finally, *Chapter 6* will present the results obtained together with the conclusions, as well as proposals to improve these results, the tools analyzed and the recommended path for the developers of these tools, the administrators who use them, and for future analyses.

It is also worth noting that, since an attempt has been made to include in any case a clear and distinguishable typography, a specific format has been followed for the lines of code and the existing commands in the memory so that they can be easily distinguished. Thus:

```
This is a line of source code
```

```
This is a portion of configuration file
```

```
$ This is a command
```

2. State of the Art

In this chapter, the reader is introduced to the current state of the HPC market and the Cloud segment within it, a brief description of what Cloud Computing is, and the advantages and challenges it proposes, highlighting the importance of automation in this type of environment and finally, the most popular tools that help in this task are presented and classified.

2.1 Current framework

Preparing on-premise[4] infrastructures has always been a long and costly process. Both the public and private sectors are involved in long processes of designing the necessary infrastructure, selection of the provider or providers, hardware implementation, system configuration, test plan and acceptance of the environment to be able to carry out the main production tasks. Many times, even, they are in the situation that, once this process is finished, partially or all of the installed technology has become obsolete and they are forced to redesign certain elements of the system or assume this loss in technological development.

In response, there is currently a decrease in investment purely in physical hardware and an increase in investment in virtual and flexible environments such as those offered by cloud environments. According to the last Intersect360 Research of the HPC market (May 2022) [Figure 1], the whole HPC market has grown 5.2% in 2021 and the segment of this market that had a higher impact was the Cloud services (cloud growth more than 85% in 2021 with respect to 2020).

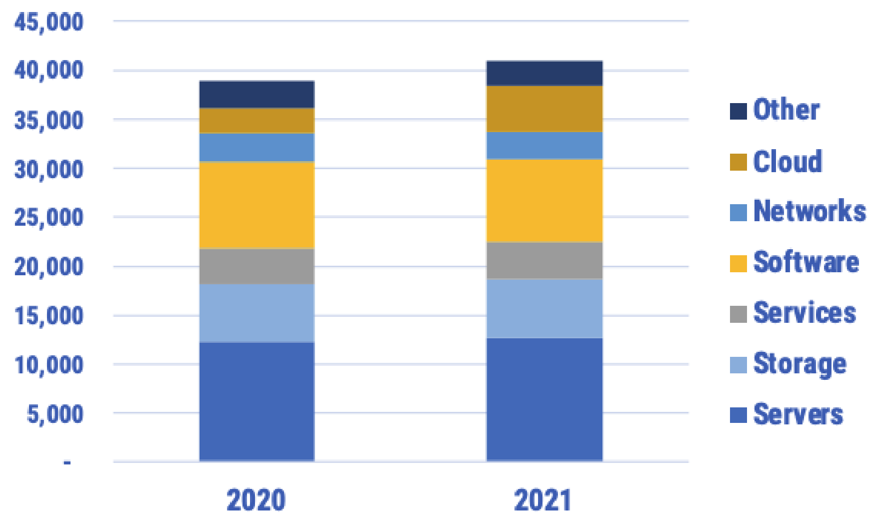


Figure 1 HPC revenue by product segment (© 2022 Intersect360 Research) [$M per year]

There are many studies on the growth of this market, its impact on society and especially its conversion to the cloud platform, considering it no longer as a complement, but as part of the infrastructure of the new HPC environments. [Figure 2] and [Figure 3] illustrates how cloud environments and services take an important

[4] On-premises refers to IT infrastructure hardware and software applications that are hosted on-site.

role in the HPC forecast for the next 4 years. This prognosis consolidates cloud environments as a small but fast-growing section of the market.

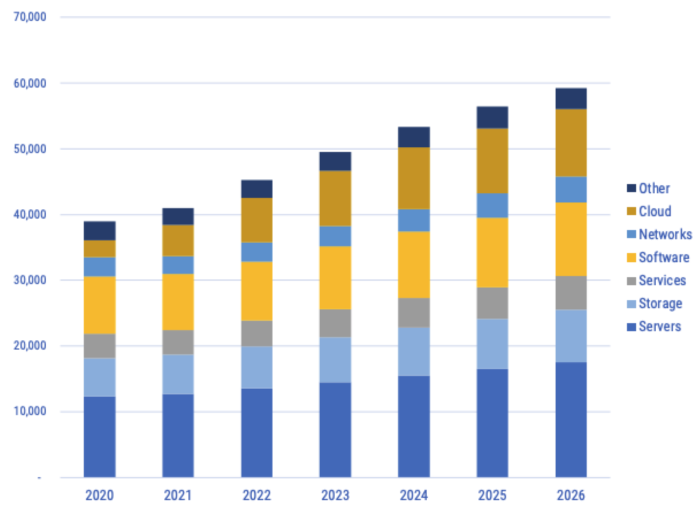


Figure 2 HPC forecast by product segment (© 2022 Intersect360 Research) [$M per year]

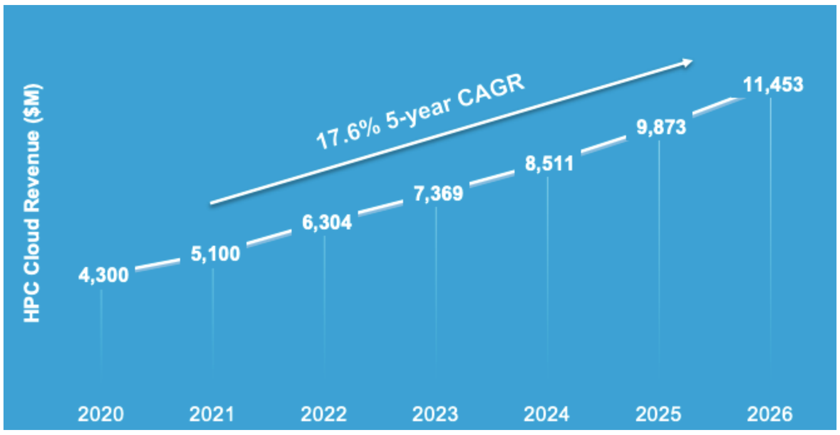


Figure 3 HPC Cloud forecast (© Hyperion Research 2022)

A clear success story of using HPC services in the cloud is the Duke University fan division project [1]. During the COVID-19 pandemic, ventilators have become a scant resource. Professor Randles and her team at Duke University found a way to split a single ventilator for two patients. Thanks to the Randles team's adoption of Azure HPC resources, they managed to run over 800,000 compute hours in just 36 hours to simulate their solution and save hundreds of lives.

The inclusion of these new modern IT infrastructures and systems derived from the cloud impact not only brings advantages such as the aforementioned flexibility and scalability, but also some factors that could be considered disadvantages such as the increase in elements in the infrastructure and therefore its complexity. These characteristics, together with the need for quick and frequent start-up and dismantling of these environments, force administrators and datacenter managers to learn new skills at breakneck speed, making it increasingly difficult to manage the required HPC workloads.

In this type of scenario, if it is not applied effective automation and standardization practice along with independence from specific cloud providers, the adoption of the multiple models for accessing the new HPC cloud becomes an almost impossible task.

Fortunately, as the scalability of HPC cloud environments concern increases, so does the recognition of the lack of trained staff in this sector and the lack of tools to ease this type of tasks, the so-called automation management tools for virtual environments.

2.2 Cloud Computing

First of all, it is vitally important to define the concepts of "Cloud" and "Cloud Computing". Although one can often be confused with the other, the truth is that they are two completely different concepts.

On the one hand, Cloud Computing is considered to be the execution in cloud environments. On the other hand, Cloud defines IT environments that pool resources and share them across a network to allow applications to run dynamically on that infrastructure.

There are many ways to classify cloud platforms, but the two most common ways are: grouping cloud types according to their location and grouping cloud types according to the service they offer.

If we consider the first definition, we can define the following cloud types:

- Public. Cloud infrastructure for open use by general public. It exists on the premises of the cloud provider (AWS, Azure, etc.) [2].
- Private. This is a cloud environment designed for exclusive use by a single organization. It may exist on or off premises and the user usually need a specific firewall or VPN access to use these resources. The OpenStack cloud of Cesga is a clear example of private cloud.
- Community. Cloud infrastructure designed for exclusive use by a specific community of users from organizations that have shared concerns. We find the European Weather Cloud [3] as an example of this kind of cloud platform.
- Hybrid. IT systems that include two or more cloud infrastructures, public or private, and that can be networked (or not). As an example, Fujitsu brand discovered the hybrid cloud benefits and offers NetApp Private Storage for Microsoft Azure.

While if we take into account the second definition, we find the following service models:

- Infrastructure as a Service (IaaS). A cloud environment that provides the capability to provision computing, storage and network resources. In this kind of scenario, the user has control over the OS and storage but limited control over network.

- Platform as a Service (PaaS). It provides programming languages, libraries, services and tools to deploy applications. Users has limited control over the platform settings to deploy the applications.
- Software as a Service (SaaS). It provides a full service accessible through the network running on a cloud infrastructure. The most common case is the Microsoft Office 365 service.

These 3 service models (IaaS, PaaS and SaaS) are considered the traditional ones in cloud environments. However, as the popularity of this platform increases, other models such as FaaS (Function as a Service), CaaS (Container as a Service) or even HPCaaS (HPC as a Service) [4] are emerging. The latter is of particular relevance in our document, since it refers to the idea of using HPC as a platform to deploy parallel and compute-intensive specific applications.

It should be noted that in no case do users have control over the underlying cloud infrastructure, otherwise the administration task would be impossible for specialized staff. Despite this limited access to a set of infrastructures, software or tools, both user researchers and administrators can find many positive aspects offered by cloud computing:

- Cost. Cloud models are based on growth on demand and payment for use. By not investing in own resources, costs can be greatly reduced in the implementation phase [5]. But the cost savings do not occur only in the initial phase in which it is decided which physical machines and how many to install, but also in the decommissioning phase, where it is essential to properly dispose of already obsolete resources. Finally, in this type of scenario, the costs can be attributed to a specific project and accounted for in more detail.
- Deployment speed. The ability to rapidly provision new clusters in a matter of minutes is key to the success of Cloud Computing, especially considering the time it typically takes to install hardware for HPC systems.
- Environmentally friendly. Conventionally, companies and organizations use their own private HPC clusters with low utilization rates with server usage spikes, deriving in negative effects for the environment. Cloud Computing contributes to and facilitates dematerialization, making data centers more efficient. It consolidates the concept of using only the resources that are needed. Therefore, the more energy efficient hardware is, the less impact on climate change.
- Flexibility. The Cloud enables rapid provisioning for certain workloads. The ability to dynamically deploy specific OS/software and hardware configurations particular to a workload is a key advantage.
- Portability. This is the ability to move applications and data from one computing environment to another. This concept enables the creation of hybrid clouds or the acceleration of adoption of cloud services with their own services. This is commonly done through some API of the cloud service.
- Scalability. this is the fundamental benefit of the Cloud. Its resilience to meet growing demands makes this feature uniquely attractive to users and administrators. Thanks to the establishment of rules and policies, administrators could reduce the number of requests by users to expand the resources that they may need at specific times.

However, as mentioned before, not all are advantages, but there are still very important challenges to cover. Among these challenges we find:

- Security. There is still a lack of trust in cloud provider, even if in some cases data is safer in the hands of public providers than in our own data center. Nevertheless, it is importante to make an effort to improve the security in this kind of platforms since data is not sufficiently secure.
- Performance. In general, the Cloud offers huge performance gains in most cases. Some workloads scale linearly, that is, embarrassingly parallel [6]. However, MPI applications may present performance issues due to I/O communications or the latency given by common network. Many traditional HPC applications are developed based on a low latency network condition. However, the main cloud providers are making a great effort to offer services of HPC systems. As a representative example, today it is possible to use infiniband networks in Azure virtual instances, something that was impossible a few years ago. Another example of this is the increasing proximity of cloud regions to all parts of the world. Traditionally, the use of the Cloud for certain applications, such as HPC, was inadequate because of the very high latency due to the remoteness of the datacenters (or regions) of the Cloud. This has been solved with the expansion of cloud provider regions, and with other products that provide direct connection lines between organizations and cloud regions, such as Azure ExpressRoute.
- Management. One of the main challenges today is the management of workloads and the resources used in cloud environments. Being able to consolidate management within a single tool is critical to effective use of the Cloud for HPC. Good management, both of the platform itself and of the costs derived from it, is not an easy task, since it is a constantly changing environment. The management of available licenses for operating systems, for example, is a difficult problem to manage in the Cloud. Clearly, there is some fear when moving from a known and understood capital spending model to a pay-as-you-go model, where costs could spiral out of control. The lack of a centralized and standardized management model for the different cloud resources capable of adapting to this dynamism and that does not pose an obstacle for IT system administrators is a challenge today.

However, it should be noted that the benefits offered outweigh the challenges for most scenarios, making the Cloud an ideal platform for most users of HPC workloads.

2.3 Automatization in cloud environments

Each cloud sets its own rules and setting up an infrastructure for a specific user request is not a trivial task. In them we can find similar services with very different names but also a wide range of different services for storage, networks, applications, etc.

Infrastructure automation is important, but it takes on special importance in the cloud world, where the number of elements increases exponentially and their management varies depending on the environment with which we work. For example, although at a conceptual level the generation of an SSH key pair is the same in a private Cloud as in a public one, the way of generating it in these models is completely different, even among the different providers.

Without application programming interfaces (APIs) and tools that centralize these functions and effectively unify them, automating these environments would be virtually impossible.

Automation is a critical element of continuous integration and delivery (CI/CD)[5] tasks that eliminate manual human interaction, increasing efficiency and reducing errors. In this way, administrators do not have to develop time-consuming manual procedures and users can execute their tasks in a matter of minutes.

Two main categories have been defined: IaC tools [2.3.1] virtual managers using IaC tools [2.3.2].

2.3.1 Infrastructure as Code tools

Infrastructure as Code (IaC) [7], as its name suggests, allows to manage and deploy infrastructures with code, rather than manually. Thanks to the use of configuration files that contain the specifications of the desired system, the description, deployment and reproduction of such environments are facilitated for center managers and administrators.

The IaC allows us to manage the needs of IT infrastructure at the same time that it collaborates with standardization, while reducing the typical errors derived from manual configuration.

Among the many advantages offered by this type of tool we find:

- Cost reduction if we have resources from pay-per-use environments, since we will have greater control over the infrastructure that we are deploying or using and reducing costs deploying underused infrastructure.
- Decrease in deployment errors avoiding manual and undocumented ad-hoc changes.
- Environment reproducibility and modularity, allowing the reuse not only of the entire system but also of independent services or elements. Moreover, these tools usually include version control, which is a recommended best practice.
- Greater standardization in the structure and procedures.
- Increased implementation speed.
- More comfortable and transparent user-experience, reducing the number of tickets/incidents and communication between the user and the admin once the environment has been launched.

This coding can be implemented on any element of the IT infrastructure: operating systems, storage, services, etc., thus facilitating the management of any team in charge of deploying these environments and of their own developers or users, who may enjoy the resources faster and free of specific human errors.

These tools are usually classified into two main groups: through a declarative approach or through an imperative approach. These two options offer a high degree of flexibility on the type of deployment that is desired, thus adapting to practically any need. The declarative approach defines the desired state of the system, so a list of resources and their properties is included. Conversely, the imperative approach

[5] Method for distributing applications to customers frequently through the use of automations in the development stages.

defines the specific commands that have to be executed to achieve the desired system.

Both approaches can be run separately or combined with each other as both offer benefits depending on the needs of users or system administrators. However, it is worth mentioning that the imperative approach forces to elaborate not only a detailed plan of commands for the creation of the environment but also for its destruction, giving way to a possible error in the rollback of the changes made. This is why IaC tools prioritize the declarative approach.

The most popular IaC solutions are:

- Ansible. Open-source tool developed by RedHat, very useful for DevOps since it allows managing servers, applications and configurations in a simple and efficient way. This tool facilitates the configuration of pre-provisioned environments and the creation of new ones.
- AWS CloudFormation. This is a declarative closed source solution. This software works specifically with AWS infrastructure in mind, the margin of error is small when working with AWS.
- Azure Resource Manager (ARM). Easy to use IaC closed source tool for the Microsoft Azure cloud environment. Another vital feature of ARM is role management and resource management, two features integral to the design that make resource allocation easier.
- Chef. Procedural tool that describes the system's steps to arrive at the final state rather than describing that state. Using aptly named "Cookbooks", one can describe various processes by which one can configure a new system to the desired state.
- Puppet. It is an open-source tool that, like Chef, can integrate with any existing platform. The main difference between Puppet and Chef is that Puppet is declarative, which some consider the preferred method of IaC. As open-source software, Puppet has a large community called forge dedicated to improving and extending Puppet.
- Terraform It is a free and open-source tool created by HashiCorp. It takes the declarative approach to IaC and nowadays is the most popular one of this category along with Ansible. Users define and provide infrastructure using a proprietary language called HCL, or optionally JSON.

2.3.2 Virtual managers: Over IaC

Due to the increasing use of cloud environments and the wide variety of providers on the market, different companies and research teams have embarked on the adventure of designing tools that facilitate the management and automation of different systems in these virtual environments, as well as monitoring their status: these are the so-called virtual managers.

In fact, these applications are at first sight very interesting, since there are even some, such as Infrastructure Manager or ElastiCluster, that are specifically designed to automate, manage and monitor the deployment of virtual HPC clusters in the cloud.

If we analyze these applications, we discover that many of them make use of IaC tools in their lower layers. That is why in our analysis we decided to call them "Over IaC" tools.

Currently, there are multiple tools that allow us to manage our cloud resources and services in a simple way, removing us from any code programming or specific programming language. Some of these tools are open-source, although most are charged tools. Among the most popular we can find:

- AWS Parallel Cluster. This is an open source tool that simplifies the creation and management of HPC clusters on AWS. It supports different batch schedulers among which SLURM and different applications and familiar scientific libraries such as MPI.
- Azure CycleCloud. This is the enterprise solution for orchestration and deployment of HPC environments on Azure. It offers the commonly used schedulers such as Slurm, PBS or LSF, distributed filesystems such us BeeGFS, and support of all the included features for administrators and users.
- CloudHealth. This is a privately software that provides cloud computing services related to cost management, governance, automation, security and performance.
- Elasticluster. Open-source cloud management tool developed by the University of Zurich that provides a user-friendly command line tool to create, manage and setup computing clusters hosted on cloud infrastructures.
- Infrastructure Manager (IM). Open-source tool, developed by the Grid and High Performance Computing Group (GRyCAP) of the Institute of Instrumentation for Molecular Imaging (I3M) at the Universitat Politècnica de València, that ease the access and the usability of IaaS clouds by automating the VMI selection, deployment, configuration, software installation and management.
- Morpheus. Leading cloud application management and orchestration platform designed from the ground up for truly agnostic cloud management.
- Red Hat Satellite. This is an infrastructure management product specifically designed to keep Red Hat Enterprise Linux environments and other Red Hat infrastructure running efficiently, secure, and compliant with various standards.

These types of tools offer multiple advantages over the direct use of IaC tools, such as greater control over infrastructure security, a graphical environment from which resources can be easily monitored and managed, and greater abstraction over the different cloud environments and underlying software.

3. Experiment: IaC vs Over IaC

This chapter describes the first contact with some tools for automating HPC cluster deployments in cloud environments. Two concrete cases of deployment attempts with the so-called "Over IaC" tools and the conclusion of this first experiment are presented.

3.1 Deployment of Infrastructure Manager (IM)

Our first approach to virtual infrastructure management and automation tools in cloud environments has been to install the IM (Infrastructure Manager) tool. This tool was of great interest from the beginning since multiple users of Cesga make use of it but in an infrastructure supported by the development team of the Polytechnic University of Valencia. The desire to deploy this open-source tool in Cesga's own facilities was the initial motivation for this analysis.

Infrastructure Manager is a cloud-agnostic tool that allows the creation of virtual infrastructure in a wide range of cloud environments (private such as Cesga's OpenStack or public such as AWS, Azure, Google Cloud or Microsoft Azure). With IM, administrators can deploy o non-premises, public and scientific clouds through multiple interfaces: there is a CLI tool for those who prefer the command line interaction with the service but also, there is a web interface that facilitates the deployment and management of our clusters in virtual environments. Powered by Ansible, IM provides a series of predefined templates for the creation of a SLURM cluster, which is of special interest in our analysis, or other parallel computing clusters such as Hadoop or Spark, in addition to being able to develop our own templates. All these features offer high flexibility in the development of new virtual infrastructures, without dependence on a specific vendor or a specific software solution, making it a highly attractive tool for managing dynamic virtual environments.

IM allows users to create and destroy new infrastructures, but also it allows to get current information and modify some features of the existing resources. All the infrastructure can be described using OASIS TOSCA Simple Profile in YAML [8] or the native RADL language.

The web-based administration interface is very attractive when it comes to deploying a tool that facilitates the adoption of this type of hybrid environments for non-IT users.

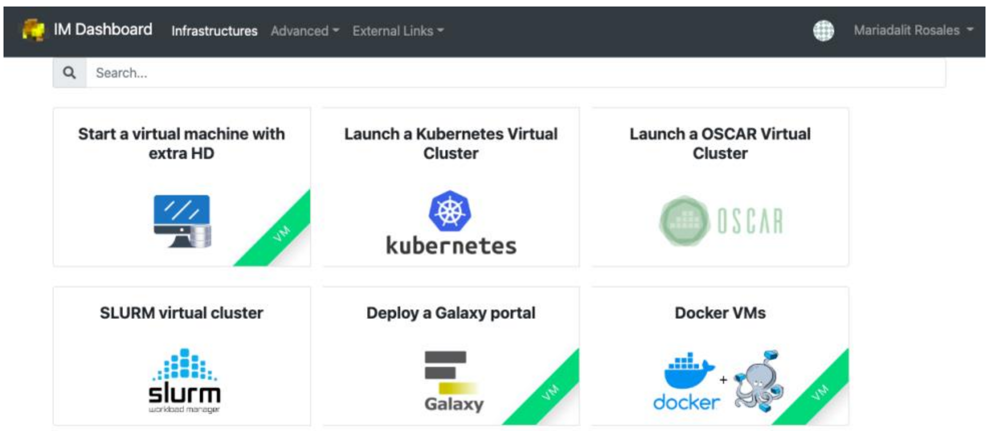


Figure 4 IM Dashboard with predefined templates

However, when evaluating this or other tools, we must not only take into account the user experience but also its management by administrators and the viability of its installation and continuous development.

In the first instance, this tool offers multiple installation media. There are docker containers prepared for each of the components of the tool, although if the administrator wishes, he can do a manual installation step by step. Whether we opt for one way or the other, the administrator encounters various problems in the tool's installation phase itself.

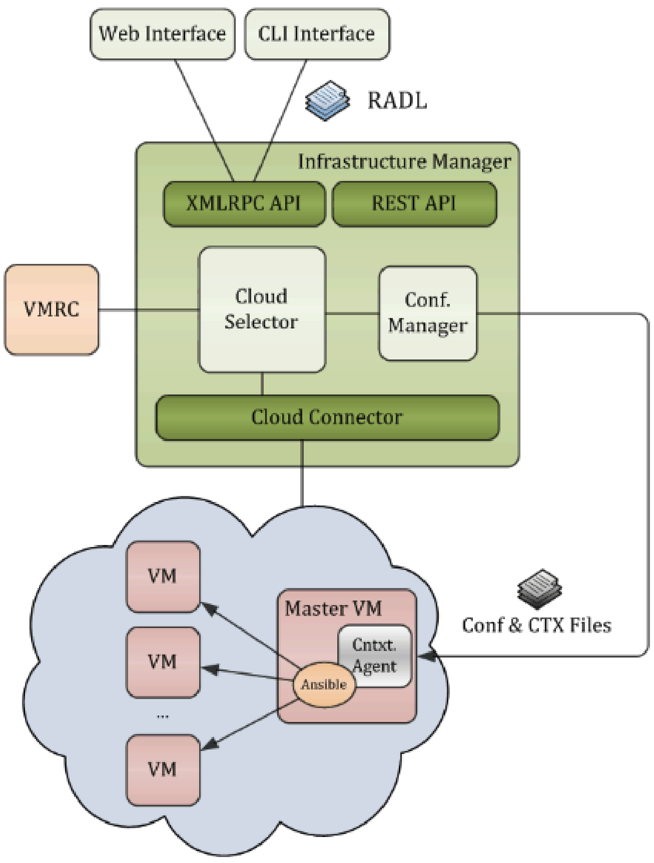
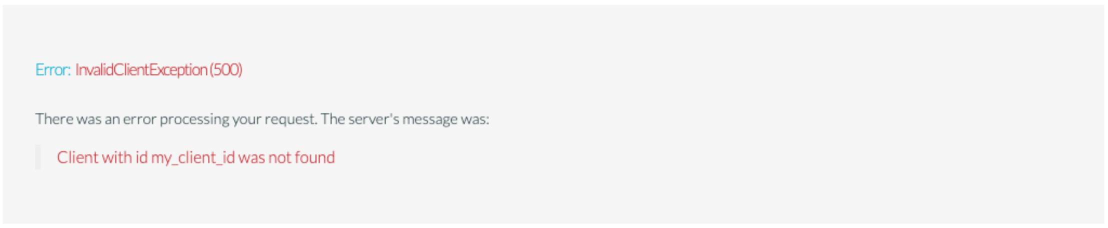


Figure 5 IM infrastructure diagram

As seen in the previous diagram, the installation of the complete IM service depends on multiple subelements. The installation through docker is made up of the execution of 3 containers: the container with the IM server, a container with an IM client that would allow us to use the tool through the command line and a container with a web portal that facilitates interaction with the tool. Our main interest lies in this web portal, since there are not many open-source tools on the market that offer a graphical interface that facilitates the creation and management of virtual clusters in the Cloud. Despite lifting all the containers and checking the services step by step as it appears in the official documentation, it is not until the end of it that we discover that the most modern and improved panel of the web portal has been developed, which is the one that is in use in the production environment offered to different research teams such as Cesga. The deployment of this new dashboard is also prepared in a docker container, but it has blocking requirements for a large number of those interested in deploying the tool. First of all, the authentication of this dashboard is limited to the OpenID-Connect protocol. In the event that it is desired to implement the tool with another authentication model such as LDAP, it would force administrators to change the tool's own code, thus hindering its early and comfortable adoption.


Error: InvalidClientException (500)

There was an error processing your request. The server's message was:

Client with id my_client_id was not found

Figure 6 OpenID-Connect authentication error in IM Dashboard

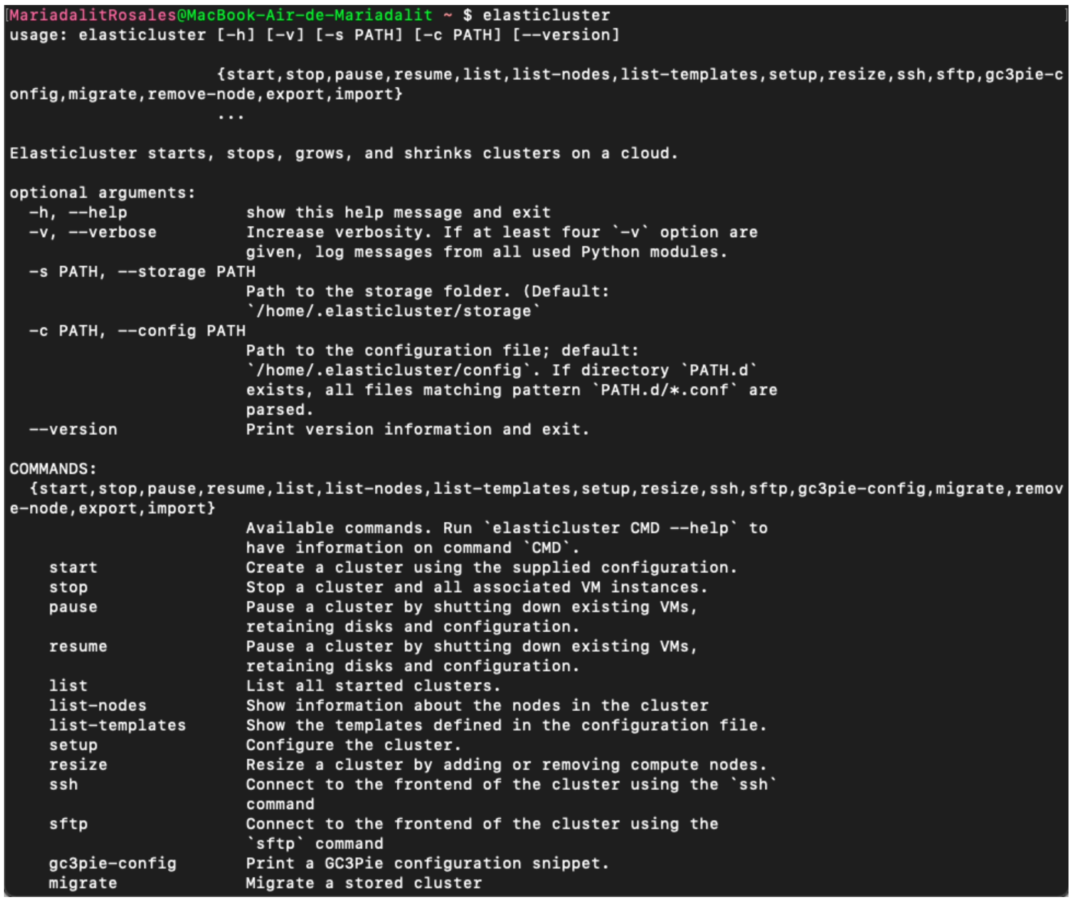
On the other hand, the documentation refers to the basic dependencies of the tool for a manual installation, but the minimum resource requirements or steps to follow to carry out the installation of these dependencies and subsequently of the tool itself are not commented on the official documentation. For a successful manual installation, it has been necessary to deploy the first installation model, that is, the docker containers, access these containers and observe the system configuration, dependencies and the configuration of the service itself to be able to execute it manually. The lack of detail in the documentation about the manual process, especially in the dependencies section, makes this installation model another difficult path for administrators to adopt.

The complexity in the deployment phase of said tool derived from the scarce documentation in this regard and the difficulty in altering the authentication method leads us to search for equivalent tools for the automatic deployment of HPC systems.

3.2 Deployment of ElastiCluster

Given the complexity and blocking points encountered with IM, the next virtual environment management tool that was considered is ElastiCluster, which, like IM, seeks to provide a more comfortable way to create and manage our virtual clusters in cloud environments. This tool has a similar set of characteristics to IM: is also cloud-agnostic since it allows us to connect to a varied group of cloud environments (OpenStack, AWS, Google Cloud, Microsoft Azure or Apache libcloud), although more restricted than IM. This tool also has predefined templates for creating clusters with SLURM, Spark or Hadoop.

The main disadvantage of Elasticluster compared to IM is that it does not have a graphical environment that facilitates the management of virtual clusters, an environment that is especially interesting for users to reduce their time-to-deploy. However, if the installation model is simple, it has similarities to the templates given with the tool, and integrating it into a new environment is not a big hurdle, the disadvantage of not having such a web portal can be compensated.



```
MariadalitRosales@MacBook-Air-de-Mariadalit ~ $ elasticluster
usage: elasticluster [-h] [-v] [-s PATH] [-c PATH] [--version]

                     {start,stop,pause,resume,list,list-nodes,list-templates,setup,resize,ssh,sftp,gc3pie-c
onfig,migrate,remove-node,export,import}
                     ...

Elasticluster starts, stops, grows, and shrinks clusters on a cloud.

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Increase verbosity. If at least four `-v` option are
                        given, log messages from all used Python modules.
  -s PATH, --storage PATH
                        Path to the storage folder. (Default:
                        `/home/.elasticluster/storage`
  -c PATH, --config PATH
                        Path to the configuration file; default:
                        `/home/.elasticluster/config`. If directory `PATH.d`
                        exists, all files matching pattern `PATH.d/*.conf` are
                        parsed.
  --version             Print version information and exit.

COMMANDS:
  {start,stop,pause,resume,list,list-nodes,list-templates,setup,resize,ssh,sftp,gc3pie-config,migrate,remov
e-node,export,import}
                        Available commands. Run `elasticluster CMD --help` to
                        have information on command `CMD`.
    start               Create a cluster using the supplied configuration.
    stop                Stop a cluster and all associated VM instances.
    pause               Pause a cluster by shutting down existing VMs,
                        retaining disks and configuration.
    resume              Pause a cluster by shutting down existing VMs,
                        retaining disks and configuration.
    list                List all started clusters.
    list-nodes          Show information about the nodes in the cluster
    list-templates      Show the templates defined in the configuration file.
    setup               Configure the cluster.
    resize              Resize a cluster by adding or removing compute nodes.
    ssh                 Connect to the frontend of the cluster using the `ssh`
                        command
    sftp                Connect to the frontend of the cluster using the
                        `sftp` command
    gc3pie-config       Print a GC3Pie configuration snippet.
    migrate             Migrate a stored cluster
```

Figure 7 Elasticluster command line tool usage

Elasticluster also has two installation modes: a quick one based on launching a docker container, and a manual installation based on installing Python modules. Both installation modes have been tested and have not posed any limitations to analyze the tool. However, little flexibility was detected in the tool since it has predefined templates to create a cluster with SLURM, but when it comes to deploying a basic HPC cluster, that is, with SLURM and some very few package requirements to be able to launch parallel applications, the need to deploy custom templates is encountered. For instance, the following configuration provides an HPC cluster with SLURM batch scheduler, but there is no possibility to add directly with this template common scientific software such as MPICH or OpenBLAS or with more specific network requiremens such as an NFS server.

```
[cloud/openstack]
provider=openstack
auth_url=https://cloud.srv.cesga.es:5000
project_name=hpc-project-uscecmrr
username=uscecmrr
password=xxxxxx
user_domain_name=hpc

[login/ubuntu]
image_user=cesgaxuser
image_sudo=True
user_key_name=elastic
user_key_private=~/.ssh/id_rsa
```

```
user_key_public=~/.ssh/id_rsa.pub

[setup/slurm]
provider=ansible
master_groups=slurm_master
worker_groups=slurm_worker

[cluster/slurm]
setup=slurm
master_nodes=1
worker_nodes=1
ssh_to=master
cloud=openstack
flavor=m1.medium
network_ids=13629491-a208-4772-b9f9-5c217177bb6fsecurity_group=default

# Ubuntu
image_id=0d8d7e8c-2cde-4cba-8a92-b56493b30cb8

# `login` info is -in theory- image-specific
login=ubuntu
```

When analyzing the possibility of creating a template that meets the objective of deploying the basic HPC cluster described, it is observed that, in fact, ElastiCluster uses Ansible to perform the deployments in the clouds that are configured. These templates, far from being documented, present problems to be customized, since they inherit many variables that serve to service the frontend that the application presents to the user via command line and that in theory serves to facilitate the operation with the deployment.

3.3 Results achieved

After this first experiment on the feasibility of installing and using these virtual managers (IM and Elasticluster), of great interest due to the supposed abstraction by administrators and users on the different platforms and underlying technologies, we have discovered that these tools are still in their maturity phase to be able to use them comfortably and quickly, facilitating their adoption and standardization in the environments that concern us.

In particular, due to the difficulty of installing IM or the lack of flexibility of the ElastiCluster templates, it is considered appropriate to analyze the underlying tools used by these Virtual Managers, the so-called IaC, to facilitate the adoption of some of these tools until that virtual managers reach an opportune level of maturity.

Next, the evaluation process and the qualitative and quantitative results of the proposed metrics for evaluation of HPC system management tools in Cloud environments using the two most popular infrastructure-as-code tools: Terraform and Ansible are described in detail.

4. Design of the Analysis

This chapter presents the method to be followed to evaluate the selected automation tools. It describes the methodology, the tools to be evaluated, the metrics that describe the essential and additional features for an administrator and the target infrastructure to be deployed in order to discover the value of each of the selected metrics for each tool. .

4.1 Methodology

After having defined the theoretical framework and the state of the art, and having carried out a small experiment in which the capabilities of the "over IaC" tools are analyzed, it is time to establish the method to carry out the comparative analysis of the management tools of infrastructures that allow us to carry out an automatic deployment of an HPC cluster.

For the analysis, an experimental methodology will be followed, in which two tools for the automatic management and deployment of a virtual HPC cluster in a cloud environment will be tested, taking measurements and analyzing the capabilities of the tools when managing and performing different tasks that are supposed to be necessary.

Broadly speaking, the experiment will consist of deploying a prototype HPC cluster in the private cloud available at Cesga with a specific infrastructure that will be defined later in the corresponding section. To achieve this goal, it will be necessary to install and configure the analyzed tools, which will be done on a controlled machine and external to Cesga environment. During this process, different qualitative and quantitative data will be taken according to the specific metrics that will be defined in the following section. A classification of the metrics to be analyzed will be created, which in turn will be divided into different evaluable items. Thus, characteristics such as the resources required by each application, the portability of the system, the scalability and the confidence in the correct functioning of the tool will play an important role in the decision. The lack of basic system resources such as CPU or RAM lead to the impossibility of deploying the desired virtual environments. The creation time of these environments and the confidence in a system of scalability on demand is considered part of the service itself and fundamental advantages of using cloud environments. If the creation times of the HPC cluster automatically exceed the times of a manual creation or if there is no possibility of adding new computing nodes due to the increase in demand by the user, these tools lose great appeal for administrators. Other additional features such as the complexity of use, the maintenance of the tool or the helplessness on the part of the provider to cover new security holes found can make it difficult to adopt it.

Finally, the data collected will be compiled in a comparative table to present the final results of the experiment.

4.2 Tools to be analyzed

After conducting an experiment using two virtual managers or two "over IaC" tools, it became clear that this technology is effectively at a very early stage of its development and that its poor documentation and rigidity when it comes to customizing the deployment, making the learning curve for administrators very large compared to the benefits it can offer with respect to IaC tools, which are in fact used in the sublayers of these applications. Therefore, it was finally decided to discard this "Over IaC" tools and focus the analysis directly on IaC tools, so in this section the two tools that will be part of the final feasibility analysis will be chosen.

As has been mentioned in previous sections, the most popular IaC tools are Puppet, Chef, Terraform, Ansible, AWS CloudFormation and Azure Resource Manager. On the one hand, AWS CloudFormation and Azure Resource Manager become very useful when working with their own cloud environments, but the impossibility of using these tools in other environments such as a private cluster with OpenStack like the one Cesga has, makes the use of these tools very limited in multiplatform environments and in particular in the scenario that concerns us. On the other hand, the Puppet and Chef tools have lost great popularity in recent years due to their client-server model, that forces to install an agent on the side you want to configure. On the contrary, Terraform and Ansible have become the most popular tools for deploying virtual systems from scratch in cloud environments since they do not need to be installed in the environments to be configured and offer multiple features to improve automation and daily administration tasks.

The following image shows a popularity ranking of the above tools based on Google search statistics:

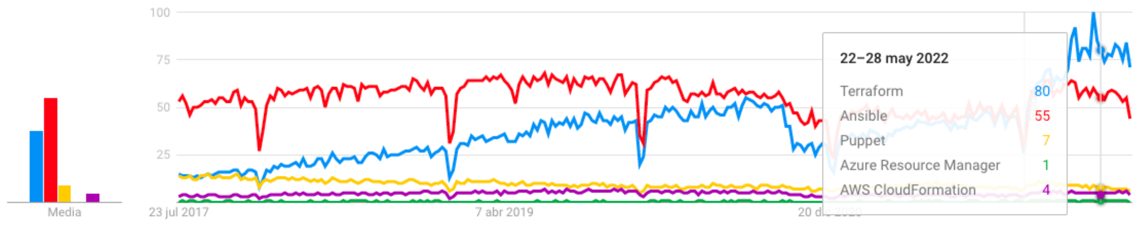


Figure 8 Interest in the last 5 years on main IaC tools (Google Trends)

Currently, there are many debates about which IaC tool offers the most advantages, focusing on the two most popular today: Terraform and Ansible. However, an analysis has not yet been done on which tool may be of greater interest to administrators of IT systems as complex as HPC systems. For this reason, in the following sections we will focus on the deployment of an HPC cluster using both tools, assessing each of the exposed metrics of interest to professionals in the IT sector and offering an objective comparison that facilitates the decision to adopt one of them for those professionals who can count on the advantages of a cloud environment for the deployment of parallel computing systems such as a classic HPC system.

4.3 Evaluation metrics

Once the method that will be used to compare the tools has been established, and the experiment and tests have been defined, it is time to establish what is going to be measured. The objective is to define a series of metrics [9] that allow to establish both qualitatively and quantitatively different essential aspects of these tools that allow to automatically manage the deployment of virtual HPC environments.

To establish the metrics, different needs that the cloud HPC cluster deployment tools must cover will be considered from different points of view, since the objective of these tools is to facilitate the configuration and creation of a new cluster that will possibly be temporary and will be created and destroyed on demand. In this sense, there are essential aspects such as the adaptation of these tools to different environments, flexibility, ease of use, or performance that will make an organization, and ultimately, the administrators of these clusters, decide for one tool or another.

In order to define metrics that provide a rigorous, fair and practical comparison to help managers choose the most appropriate tool for their organization, metrics will be classified into two categories: essential features and additional features:

- **Essential features**: these are functions that are essential to achieve the goal of deploying an HPC cluster in virtual environments automatically, flexibly and efficiently.
- **Additional features**: these are functions that, although not directly focused on achieving the best deployment of an HPC cluster, are necessary for the correct operation of the tools, their management or configuration, and that ultimately facilitate the administrator's day-to-day work.

Within these two categories, the different metrics will be classified, which, in turn, will have key **aspects** to facilitate comparison. Five key aspects will be compared for the **essential metrics**, and two key aspects for the **additional metrics**.

In addition, a scoring system is established, which, although it is considered subjective, serves as a reference model for the comparison of tools. A maximum weight of 2 will be given to all the Essential features that offer a clear advantage to these tools and a weight of 1 to those additional characteristics that provide a differentiating value to each tool but are not essential since they can be covered mostly by third party tools. It is important to note that the weights can be given in a partial format. This means that for each essential aspect evaluated, 0, 1 or 2 points can be given, while for additional aspects, only the full score, or no points at all, may be provided.

Since, as described above, there will be a total of 16 key aspects for the essential metrics and 8 key aspects for the additional metrics, a total of 40 points can be obtained.

Next to each metric, the different aspects that will be evaluated will be included, and each one of them will include its description, the way it will be evaluated, and the technical resources needed to perform this evaluation. This will help to perform the analysis more efficiently in Chapter 6. Finally, all the aspects will be summarized in a table that will help to make the comparison in a simpler way for the reader.

4.3.1 Essential features

1. **Performance**

Performance is an essential feature in the deployment of an HPC cluster since administrators need to ensure that the tools they install and configure do not make excessive use of available resources. A tool that makes high RAM usage every time it is launched jeopardizes the health of the node where it is hosted. In environment automation tools it is also important that the response times do not exceed a minimum, because if the manual deployment does not significantly exceed the automatic deployment in terms of effort and time, the use of these tools will deteriorate as there is no direct benefit from their use. The following aspects will be measured:

a. Time of execution:
   - Description: Time taken by the tool for the complete the creation and the destruction of the test HPC cluster, from its launch to the moment when it is possible to start using it.
   - How to measure: The command *$ time* natively provided by UNIX will be launched with the command used by the tool to deploy the the infrastructure. This process will be executed five times and the medium value will be taken as the result. Example:

```
$ time tested_tool deploy_and_destroy_command
```

b. CPU usage during deployment.
   - Description: The possible CPU usage peaks that are encountered will be emphasized, since it is of vital importance not to seriously impact the performance of the machine on which the environment is deployed, as this will increase the deployment time, and the stability of the machine will be lower.
   - How to measure: as there is no native and easy way to get the CPU usage peak during the execution of a command, the command top will be used to get all the CPU usage values and the highest one will be selected as peak. This process will be executed five times and the medium value will be taken as the result. The following command will be used:

```
$ while true; do top -a -pid $(pidof tested_tool | cut -d " "
-f1) >> cpulogfile; done
```

c. Memory usage during deployment.
   - Description: Similar to the previous aspect, it is essential that memory usage does not spike too much during deployment, as again this will negatively impact the overall performance of the machine on which the virtual cluster is deployed.
   - How to measure: Using the *-l* option of the *time* command, the returned value contains the memory peak used by the command launched. In this example, an output is showed with the data that will be taken in bold. This process will be executed five times and the medium value will be taken as the result.

```
$ /usr/bin/time –l tested_tool deploy_command
[…]
     122,32 real        28,85 user         5,60 sys
              43909120  maximum resident set size
[…]
```

d. API calls and network usage

- **Description**: Another important aspect for performance, and also closely related to cost in some cases, is the number of cloud API calls that are being used. To perform this deployment, the tool will contact via API the cloud providers used to request different actions. This can affect network performance, and in addition, especially in public clouds, there is a limit to the number of API calls that can be made in a certain period.
- **How to measure**: To compare the same values, Wireshark and tcpdump will be used to measure the network stadistics between the host and the API URL of the cloud provider used. The traffic will be captured using the following command.

```
$ tcpdump -s 0 -w network.pcap
```

Afterwards, using Wireshark, the packets will be filtered to get the values of interest, that are the number of packets and the number of bytes. To do this, the following filter will be used inside Wireshark:

```
ip.addr==193.144.32.42
```

2. Portability

Portability could be defined as the ability of a tool to work in a new environment. An example of portable and simple software could be a Docker container, since it allows a high level of abstraction of the environment in which it runs. In this sense, this feature is defined as being of great importance as it implies the ability of the tool to work in different environments quickly and easily. In addition, this feature will help to define the necessary requirements for the compared tool to run and be able to deploy the virtual HPC cluster. The aspects to be compared will be the following:

a. Storage dependencies
   - **Description**: This aspect will look at storage dependencies, such as whether a particular type of disk is required or if it needs to use a large amount of space.
   - **How to measure**: Official minimum requirements to install the tested tool.

b. Memory dependencies
   - **Description**: Similar to the previous aspect, in this case the tool's memory dependencies, such as whether a specific type of memory is needed or the minimum requirements indicated by the manufacturer, will be analyzed.
   - **How to measure**: Official minimum requirements to install the tested tool.

c. OS dependencies
   - **Description**: In this case, the portability of the tool relative to the operating system will be analyzed in order to clarify whether the tool will be able to deploy the cluster from different operating systems, if it needs specific ones, or if it needs additional system libraries.
   - **How to measure**: Official minimum requirements to install the tested tool.

d. Third-party libraries required

- **Description**: Related to the previous aspect, it will be analyzed whether the deployment tool requires additional libraries that are not included in the operating systems, or other tools that must be installed for its correct operation.
- **How to measure**: Official minimum requirements to install the tested tool.

3. Reliability

Administrators need tools that ensure correct operation when specific functionality is required, i.e., that the number of software-derived failures is as few or non-existent as possible. This characteristic is a fundamental element to ensure reliable operational use. These are the aspects to analyze it:

a. Log information
   - **Description**: When these tools are running, it is important to have information about the actions you are performing and their result so that, if errors occur, they can be easily identified and fixed. Therefore, this aspect will evaluate the callback information provided by the tool during its execution, and the logs stored after this same execution.
   - **How to measure**: Analyzing if the log level can be modified, and other configuration available related to logs (possibility to change path, for example) together with the callback information when the tool is executed.

b. Number of failed launches
   - **Description**: In this aspect, the number of failures generated in each tool used for the comparison will be measured within a controlled environment. This measure is interesting and derives precisely from the performance of the tool: if resource consumption is high, the stability of the platform on which the deployment is performed will be reduced. In addition, if the tool requires many calls to the cloud provider's API, it may exceed the usage limit or affect the stability of the network as well.
   - **How to measure**: perform the deployment three times and take the number of failed deployments. A deployment will be considered failed if there is any kind of error during it.

c. Workflow control
   - **Description**: Within this aspect, it will be evaluated if the tool includes commands or utilities that allow modifying the operation flow. For this purpose, the existence of commands to pause, restart or interrupt the platform deployment flow will be taken into account.
   - **How to measure:** Analyze if there are commands to pause, restart or interrupt the deployment and its usage.

d. Status control
   - **Description**: This aspect will evaluate the tool's ability to provide information on the status of the deployed cluster after the deployment has been completed. This will allow the administrator to know key information about his new environment without using additional tools such as Grafana.

- How to measure: Analyze if the tested tool provides native ways to get the status of the cloud provider, the deployed resource and the instances.

4. Scalability and flexibility

Scalability and flexibility are one of the most expected features of these tools when deploying virtual HPC clusters, and due to the nature of these deployments, simplicity and speed are expected when deploying larger clusters, or including new features in a preconfigured way without further intervention by the administrator, either through proprietary or third-party plugins, or reusing previously created modules.

a. Scalability capacity
   - Description: scalability is considered to be how well each tool works as we increase the number of instances in the virtual cluster.
   - How to measure: To analyze this characteristic, the execution time will be measured when launching an HPC cluster with 1, 2 or 3 compute nodes. Again, this will be done with the *time* command as stated in metric 1, aspect a).

b. Compatibility with cloud providers
   - Description: This aspect will take into account the ability of the deployment tool to integrate with the different public and private clouds, allowing deployments using one or several of them.
   - How to measure: Analyzing the official documentation to determine if the main cloud providers are compatible, and also taking into account if additional cloud providers can be configured.

c. Capacity to integrate custom plugins
   - Description: Ability to integrate modules developed either by third parties or by the cluster management team itself in order to, for example, add functionality to the cluster in terms of pre-installed and pre-configured software.
   - How to measure: Analyzing the official documentation to determine if the tool is compatible with custom plugins or modules to perform additional actions no officially supported.

d. Modularity
   - Description: This aspect will measure the possibility of modularly reusing code from different deployments already made to add it to new ones. It will also take into account the ability to launch deployments disabling certain features.
   - How to measure: Analyzing how easy is to reuse code, or deployment configuration, in a different deployment.

4.3.2 Additional features

1. Cost

One of the main disadvantages of some tools is that they are only offered in a paid mode. Not all administrators can or wish to invest part of their funds in paid tools but prefer to collaborate with free software tools. It is therefore vital to find a reliable and useful tool, but one that does not require monetary capital for testing and implementation.

a. Open-source option
   - Description: This aspect will evaluate the existence of an open source version, the support received by the manufacturer and the support received by the community. Failing that, the free version of the tool will be evaluated. The features available in these versions that have no associated cost for the users will be taken into account.
   - How to measure: existence of open-source option and the characteristics included.

b. Enterprise version option
   - Description: This aspect would evaluate the existence of an "enterprise" version, i.e. a version with all the features, manufacturer support and no limitations. Both the costs and the additional features available will be taken into account.
   - How to measure: existence of paid version and the characteristics included compared to the free version.

2. Maintainability

One of the main obstacles often encountered by critical environment administrators is the difficulty in maintaining the installed software. As technology advances, it is essential to have a tool that has an active development, which ensures the correction of bugs or the development of new functionalities or needs discovered and requested by users. In addition, it is important that the update model of the tool is as fast and transparent as possible, to avoid any loss of service or impact to end users.

a. Documentation
   - Description: Documentation is essential to facilitate the use of a tool and reduce the learning curve. In this aspect, its quality, ease of reading and available languages will be evaluated.
   - How to measure: Analyzing the usability of the official documentation of the tool, the examples included and the community behind each tool.

b. Software releases policy
   - Description: As a critical tool, the software lifecycle policy must be transparent and clear for the administrator to control it.
   - How to measure: Analyze how each provider announces a new version and the procedure to update it.

3. Security

Cyber-attacks have multiplied in recent years, and this makes the security aspect something to consider when choosing a deployment tool for an HPC cluster. At this point it is important to highlight the criticality of these applications, since an attack involving the deployment tool affects the entire chain, up to the application executed in the cluster and also its results. These are the specific aspects that will be evaluated:

a. Vulnerability control and tracking
   - Description: Every software tool is affected at some point by vulnerabilities, but the most important part is to know how the provider communicates and follows it, by, for example, providing workarounds to solve them and how it is followed up until its solution.
   - How to measure: Analyze the lastest known vulnerability for the tested tool and how the providers give information about it.

b. Identity management
   - Description: As it is a critical tool, the incorrect or unauthorized use of the tool itself, or the modification of its configuration files, should be completely controlled.
   - How to measure: Analyze the resources provided by each of the tested tools to control permissions, users and roles to use the tool itself.

4. Complexity

It is essential that the adoption of a new tool is not an obstacle for the administrator. It is necessary to use applications that make tasks easier for them and that new members can learn how to use them easily and quickly, without requiring a great learning curve to be able to use the tool effectively. This complexity can be measured by taking into account characteristics such as the language used in the configuration or the difficulty to install and run the tool.

a. High level language and syntax check
   - Description: The complexity of a management tool is largely defined by the syntax of its executions and its configuration files. Using a common language and having syntax check options is usually the best option in all cases.
   - How to measure: Analyze the language used by each compared tool, whether it is its own or standard, as well as extra functionalities such as plugins for text editors that allow checking the syntax of the generated configuration and easily detecting errors.

b. Installation command
   - Description: A way to measure the ease of use of a tool is how it is installed and the dependencies needed.
   - How to measure: In this aspect, the number of commands necessary to carry out the complete installation of the tool within a controlled environment will be measured.

In order to make the comparison more visual, the analysis of the different aspects will be summarized in the following table.

Essential features		
Performance		
Aspect	Comments	Points
Time of execution		
CPU usage during deployment		
Memory usage during deployment		
API calls and network usage		
Portability		
Aspect	Comments	Points
Storage dependencies		
Memory dependencies		
OS dependencies		
Third-party libraries required		
Reliability		
Aspect	Comments	Points
Log information		
Number of failed launches		
Workflow control		
Status control		
Scalability and flexibility		
Aspect	Comments	Points
Scalability capacity		
Compatibility with cloud providers		
Capacity to integrate custom plugins		
Modularity		

Table 1 Essential features reference

Additional features		
Cost		
Aspect	Comments	Points
Open-source option		
Enterprise version option		
Maintainability		
Aspect	Comments	Points
Documentation		
Software releases policy		
Security		
Aspect	Comments	Points
Vulnerability control and tracking		
Identity management		
Complexity		
Aspect	Comments	Points
High level language and syntax check		
Installation command		

Table 2 Additional features reference

4.4 Prototype infrastructure

It has been decided to use as a prototype infrastructure an HPC cluster made up of virtual instances within the OpenStack private cloud provided by Cesga.

There are many elements and features that define an HPC cluster. However, for our analysis, we have decided to deploy a simple HPC cluster that has all the minimum components and tools to run a pair of parallel test codes that compute the pi number.

```
# This code has been obtained from
# https://gist.github.com/Macorreag/3bd78ee36a5c2d4141e9135ceae31b77
#include <stdio.h>
#include <string.h>
#include <mpi.h>
#include <omp.h>
#include <math.h>

#define ITERATIONS 2e09
#define MAXTHREADS 32

int calculatePi(double *pi, int ID, int numprocs)
{   int start, end;
    start = (ITERATIONS/numprocs) * ID;
    end = (ITERATIONS/numprocs) + 1;
    int i = start;

    do{
        *pi = *pi + (double)(4.0 / ((i*2)+1));
        i++;
        *pi = *pi - (double)(4.0 / ((i*2)+1));
        i++;
    }while(i < end);

    return 0;
}

int main(int argc, char *argv[])
{
      int done = 0, n, processId, numprocs, I, rc, i;
      double PI25DT = 3.141592653589793238462643;
      double local_pi[MAXTHREADS], global_pi;
      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
      MPI_Comm_rank(MPI_COMM_WORLD, &processId);
  if (processId == 0) printf("\nLaunching with %i processes",
numprocs);
  global_pi = 0.0;

  #pragma omp parallel num_threads(4)
  {
    int threadId = omp_get_thread_num();
    int threadsTotal = omp_get_num_threads();
    int globalId = (processId * threadsTotal) + threadId;
    calculatePi(&local_pi[threadId], globalId, threadsTotal*numprocs);
    #pragma omp single
    {
      for(i = 0; i < threadsTotal; i++)
        global_pi = global_pi + local_pi[i];
    }
    printf("%i ", globalId); fflush(stdout);
  }
  MPI_Reduce(local_pi, &global_pi, 1, MPI_DOUBLE, MPI_SUM, 0, 
MPI_COMM_WORLD);
      if (processId == 0) printf("\npi is approximately %.16f, Error 
is %.16f\n", global_pi, fabs(global_pi - PI25DT));
      MPI_Finalize();
      return 0;
}
```

Our reference parallel code, as in most of the applications implemented in high performance computing clusters, is developed taking into account two fundamental standards in parallel programming: OpenMP and MPI. On the one hand, the objective of OpenMP is to provide parallelism capacity within the machine itself thanks to the shared memory model. Parallelization is performed through directives, libraries and environment variables, while on the other hand, MPI provides parallelism between different nodes of the same cluster thanks to the distributed memory model through message passing.

Therefore, we have considered that our reference HPC cluster must include these libraries for parallel programming.

Moreover, if HPC users were executing their HPC jobs without any control, they would disturb each other and none would get any performance. There are multiple advantages of using a software tool called batch scheduler in the HPC systems. This kind of software schedule the execution of jobs according to rules defined by administrators and users and ensures the best optimization of resources for our applications and minimizes the execution errors. It also enables monitoring features to control the state and output of our launches. There are many popular batch schedulers like SLURM, OpenPBS, IBM Spectrum LSF and SGE. We have considered to install Slurm as a resource manager in our prototype cluster because it is the most popular in HPC systems today. Specifically, more than 60% of the supercomputers that make up the TOP500 list have SLURM installed.

The cloud resources for the deployment of this system are provided by Cesga private cloud based on OpenStack technology. Although it is true that OpenStack lacks the tools available in other cloud environments such as AWS or Azure, OpenStack offers us all the resources we need to deploy the designed HPC infrastructure, as well as offering clear advantages for the scientific and research sector, such as a greater flexibility of customization since it is an open-source software.

Our OpenStack account has a maximum of:

- 1 floating IP[6].
- 8 GB of RAM and 8 virtual CPUs.
- 4 instances running in parallel.
- 160GB of storage.
- 10 routers and 100 networks.

On top of that, in this OpenStack cloud we have 7 types of predefined instances with very specific characteristics in terms of the total number of virtual CPUs, RAM and

[6] Floating IP is a public, static IP address for instances created only in a private subnet (that is, without a public network interface). Using floating IP, such an instance will be able to accept incoming connections from the Internet.

disk. We also have a predefined list of images for our virtual instances, including multiple Linux distributions.

Taking into account the resources that we have and regarding the architecture, we have considered the simplest possible system with a single frontend node and a compute node for the measurement of all metrics, except for scalability, in which we consider the creation of an HPC cluster with more than one compute node, thus allowing us to obtain the creation and adaptation times of the tool in the face of the increase in nodes according to demand. This frontend node is responsible for performing the login and controller tasks. This node serves as an interface between the user and the cluster itself. It is common for computing nodes to be found in a private network, so this node not only serves as an optimal interface between different networks but also avoids any direct interaction between the user and the computing resources themselves. It will also perform the functions of orchestrator of the services and resources of the cluster. The compute node is the one that carry out the computing tasks. This node is limited to running applications and direct interaction with it is designed to be as little as possible. For both type of instances, we have selected the m1.tiny instance type, which has 2 vCPUs, 2 GB of RAM and 10 GB of disk and the baseos-CentOS 8 image.

It is true that the most popular HPC systems have many more nodes and other features, such as a dedicated network with low latency to speed up the communication for the passage of messages between the different machines and a much broader software offer that adapts to the user needs. However, this system is considered sufficient to represent the simplest execution of a parallel application run in a common HPC system and allows us to evaluate the essential features of the tools that concern us in this analysis in order to make it easier for administrators the decision whether or not to adopt any of them.

These nodes communicate through a private network, which is created from scratch in the automation. In order to access the frontend node, it is also necessary to create a router between the pre-existing public network and the new private network. In addition to this, we need to assigning a floating IP to the frontend and generate a pair of SSH keys to be able to access this node via SSH, since the configuration by default does not allow an SSH connection model only by password. No low latency network is tested but NFS shared folder is deployed to share configurations and output executions between instances.

As a summary at software level, the cluster nodes have installed:

- The SLURM resource manager. It requires also the installation of the MUNGE software as the authentication tool inside the cluster between nodes.
- Minimum software for launching parallel applications: C and fortran compilers, openMP libraries and OpenMPI packages.
- NFS shared directory to communicate between nodes and facilitate the transfer of execution results in the cluster.

The [Figure 9] represents the architecture of the prototype HPC cluster infrastructure to be deployed.

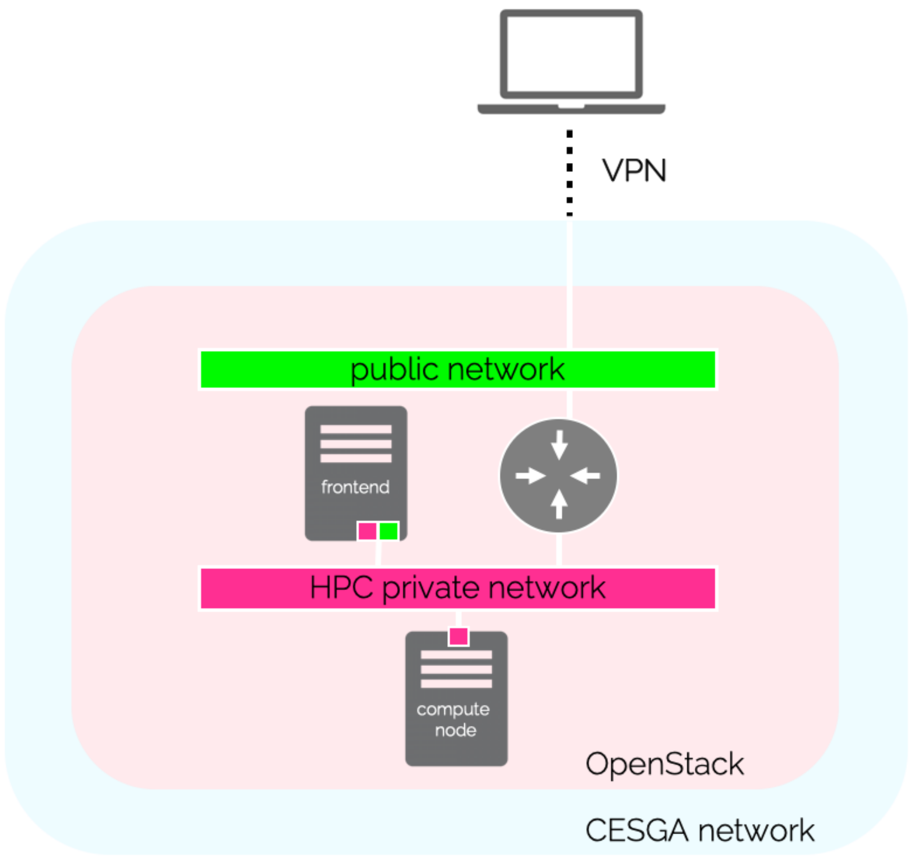


Figure 9 HPC cluster prototype architecture

The creation lifecycle of this environment has the following phases:

1. Create the SSH key pair. Whether we create a cluster with 2 nodes or 200, we will always need to generate a pair of SSH keys to be able to connect in the most secure way possible to the node that acts as a gateway between the public network and the private network of our cluster. In our scenario, the SSH key pair is generated manually executing the `ssh-keygen` command. By default, the public and private keys are generated under the ssh folder in our home directory, i.e., ~/.ssh/. Public key can be differentiated since it has the extension .pub. Once we have the keys, the public key must be uploaded to our cloud to be used for the nodes of our clusters.
2. Creation of a private network. The number of v4-type IPs available makes it impossible to assign a floating IP for each computing node. Also, for security, our HPC system should be as isolated as possible from the public network. Therefore, we need to create a virtual private network in which we can connect all our instances and thus allow direct communication between them. Subnet 10.1.0.0/24 is used as private network for the cluster performed with Terraform and 10.2.0.0/24 is used as private network for the Ansible deployed HPC cluster.
3. Creation of a virtual router. This router will be between the public network and the predefined private network to redirect traffic in both directions. This element is mandatory in order to access our frontend and thus our cluster from through Internet.
4. Frontend creation. It will be assigned a floating IP to this instance to be able to access it through the router via SSH with a public IP and we will connect it with the newly created private network to allow direct communication with the

computing node. Also a static private IP is assigned to this node to simplify the configuration of the NFS server service. After the instance creation, a post configuration script is executed at boot to perform the particular configuration of this node.

5. Creation of the computing node. The definition of this resource is almost equivalent to the frontend one but without any specific IP. We will attach this instance to the private network and its configuration will be done at first boot time by a specific script.

Phases 4 and 5 are done in parallel, while phases 2 and 3 are done sequentially since the creation of one depends on the other. Phase 1 is only a requirement for phases 4 and 5, so the moment of its execution is indifferent as long as it is carried out before said phases. Likewise, it will be the first to execute when using the deployment automation tools since it does not depend on phases 2 and 3.

Although some of the tools offer a native way to post-configure our nodes, it has been decided to use the user_data directive of our private cloud API, available in each of the IaC tools evaluated, to maintain equivalence in the comparison. This user_data parameter allows us to specify a script for each type of instance which will include the software installation and dependencies to establish a cluster with SLURM, a shared NFS directory, and a series of scientific libraries and packages for the launching of parallel applications, as previously mentioned.

As can be appreciated, the benefits of this cluster are minimal since it only has one computing node and the memory and CPU resources, together with the software available to users, are limited. Even though, its configuration is adequate to evaluate automation tools on the deployment of common HPC clusters.

5. Analysis

Terraform and Ansible are the most popular tools within the IaC tools categorization. Historically, Ansible has always been more prevalent in the IT industry due to its ease of use and widespread documentation. However, in the last couple of years, Terraform has attracted more and more interest, especially due to its ease of deployment and the increasing popularity of cloud environments.

There are now many discussions about which tool can be the most useful for the day-to-day work of administrators and center managers, especially when the administration environment becomes complex with on-premise and cloud environments. However, this debate has not yet been focused on the HPC sector and its day-to-day tasks.

In order to perform the analysis, this chapter will be divided into two main sections: one dedicated to Terraform, and the other dedicated to Ansible. Within each of these sections, the technology will be introduced, emphasizing its architecture, capabilities and mode of operation. Later, the necessary steps for its installation and configuration will be defined, with the objective of deploying the prototype HPC cluster defined in section 4.4. After testing, the tools will be evaluated according to the criteria defined in section 4.3. Finally, they will be summarized in the comparison table.

As will be indicated, it is necessary to use a computer in which both tools are installed, the configurations are made and from which each tool can orchestrate the deployment of the HPC cluster.

Operating System	MacOS Monterey v12.3.1
Processor	1.6GHz Intel Core i5 dual core
Memory	4 GB 1600 MHz DDR3

Table 3 Physical host characteristics

Although it is not of great relevance at a comparative level (since the same equipment will be used in both cases), the technical characteristics of the equipment are indicated in the table above.

It is also important to note that the analysis assumes that you have a cloud provider with the appropriate resources (Cesga's OpenStack, in this case), a user with appropriate permissions and API access to the provider.

5.1 Terraform

5.1.1 Introduction to Terraform

The Terraform project is an open-source project developed in Go by the company HashiCorp. It allows users to configure infrastructure in a high-level language, generating an execution plan to deploy it in private clouds, public clouds, or containers such as Docker. It offers support for a wide range of virtualization service providers such as OpenStack, Docker, AWS, Azure, Google Cloud, and services such as Cloudflare, RabbitMQ and Grafana.

The infrastructure is defined using a proprietary configuration syntax HCL (HashiCorp Configuration Language) or otherwise in JSON format. Files using the HCL syntax must be created with the extension *".tf"*, while files in JSON format must have the extension *"tf.json"*.

The HCL language is based on encoding the providers' standardized APIs in declarative configuration files. This declarative model allows Terraform to know the state of the manipulated infrastructure to map real world resources to internal configuration and keep track of metadata. This state is stored by default in a local file named *terraform.tfstate* so the Terraform knowledge about the infrastructure depends on this state. If a third-party modifies this state, Terraform idempotency will be deleted.

Its simple syntax together with extensive documentation make it easy to use this tool for infrastructure provisioning in cloud environments. Terraform has more than 100 providers, among which we find OpenStack, the private cloud on which we will deploy our HPC clusters.

Apparently, one of the main advantages of Terraform is that it calculates dependencies between resources automatically. However, depending on the provisioner, we have found that this is not always the case. In our particular case, we have detected that there is a race condition [10] between the private network to be created and the frontend and compute instances that depend on it, having to explicitly explain said dependency with the depends_on directive.

The open-source code repository is accessible from GitHub [11] and has an issues tracker [12] reported by HashiCorp and the community that supports it.

Terraform has two main components:

- Core. It uses two input sources: 1) Terraform configurations, where users and administrators define what need to be created or provisioned and 2) the state of those resource. Terraform core takes those inputs and figures out the plan of what needs to be done. To do so, first it compares the current state with the desired state and if something needs to be done it performs all the needed tasks to get the desired platform.
- Providers. A provider is a plugin that allows users to manage an external API. We can find cloud providers (IaaS) but also a set of providers for high level technologies such as Kubernetes (PaaS) or Fastly (SaaS).

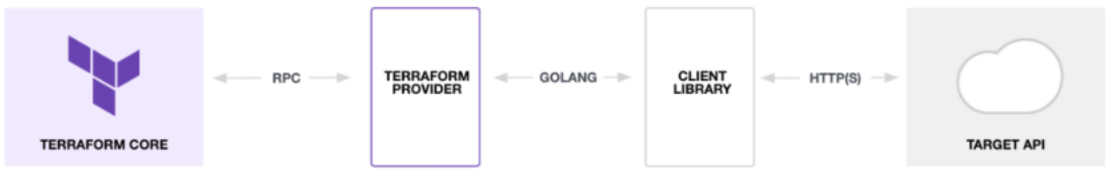


Figure 10 Terraform main architecture

Terraform has a very special component that gives more flexibility to administrators and developers that is called module. A module in Terraform is a set of configuration files inside a directory. These modules encapsulate groups of resources dedicated to one task, reducing the amount of code that needs to be developed for similar infrastructure component.

Terraform is available in different installation modes. On the one hand, they have an open-source version, which they describe as a free but self-managing tool, and two paid versions: an Enterprise version self-hosted by the enterprises and organizations that includes multiple advanced features and the cloud version, hosted by Terraform but with all advantages of Terraform Enterprise. One of the main advantages of the payment method is that it includes a graphical interface to facilitate the use of the tool, while the open-source model only includes CLI commands. However, in this analysis the open-source model will be used since it is the only free version and allows us to assess the fundamental elements of said tool.

Terraform is available for binary download or direct download using the various package managers for multiple operating systems (MacOS, Windows, Linux, FreeBSD, OpenBSD, and Solaris).

5.1.2 Configuration and cluster deployment

The first step is to install Terraform in the physical host. As mentioned above, there are multiple ways to install it, but in this case the pre-compiled binary will be used to install Terraform in this case. To do so, it is needed to download the specific package for the system, MacOS in this case, unzip de package and copy the binary named "*terraform*" to the */usr/local/bin* folder for the autocompletion.

```
$ wget
https://releases.hashicorp.com/terraform/1.2.5/terraform_1.2.5_darwin_
amd64.zip
$ unzip terraform_1.2.5_darwin_amd64.zip
$ mv terraform /usr/local/bin/
```

The installation can be tested with the following command:

```
$ terraform -help
```

Once the tool is installed, there are no additional steps, so the configuration for the deployment of the virtual HPC cluster can be started. To do this, is recommended to create a new directory where the configuration files will be stored. These files will contain the infrastructure to be deployed and the most important one is "*main.tf*".

```
$ mkdir terraform_openstack
$ touch terraform_openstack/main.tf
```

In this configuration file, the following is needed:

1. Information about the cloud provider. The name of the provider must be defined, together with some basic information. This includes the cloud account to be used through the corresponding API with its username and password, and other information such as the domain, project name or tenant name and the URL for the authentication API. Additional APIs (such as a network API) will be included in the "*endpoint_overrides*" section.

```
provider "openstack" {
  user_name   = "uscecmrr"
  tenant_name = "hpc-project-uscecmrr"
  domain_name = "hpc"
  password    = "xxxxxxx"
  auth_url    = "https://cloud.srv.cesga.es:5000"
  endpoint_overrides = {
    "Network"  = "https://cloud.srv.cesga.es:9696"
  }
}
```

2. The SSH key pair. As defined in the prototype infrastructure, SSH key pair authentication will be used between the tool (Terraform) and the different hosts that make up the virtual HPC cluster. Openstack will need the value of the public key to be able to authenticate Terraform with the machines, something that is configured in the resource type "*openstack_compute_keypair_v2*":

```
resource "openstack_compute_keypair_v2" "keypair" {
  name       = "terraform"
  public_key = "ssh-rsa *********************************"
}
```

3. The private network to connect all cluster instances. This private network will be created by Terraform. To define it, the needed parameters are: the name of this new network, the network ID (this is an internal ID used by Terraform and assigned to the resource), the CIDR and the IP version that will be used. In this case, the subnet 10.1.0.0/24 will be used for the HPC cluster deployment with Terraform. Additionally, the parameter "*admin_state_up*" defines if this subnet will be available when deployed.

```
resource "openstack_networking_network_v2" "network_private" {
  name           = "private_terraform"
  admin_state_up = "true"
}

resource "openstack_networking_subnet_v2" "subnet_private" {
  name       = "subnet_terraform"
  network_id = "${openstack_networking_network_v2.network_private.id}"
  cidr       = "10.1.0.0/24"
  ip_version = 4
}
```

4. The router. As stated, this router will redirect traffic from public network to the private network and viceversa. The router is defined within a resource of type "*openstack_networking_router_v2*". This resource will be available when deployed ("*admin_state_up*"). To define the private subnet that will be connected to the router, another resource of type "*interface*" is configured. This interface will contain the "*router*" and the "*subnet*" to be connected. This is done through their IDs, that are provided using variables.

```
resource "openstack_networking_router_v2" "router" {
  name                = "terraform_router"
  admin_state_up      = true
  external_network_id = "88ef59cb-08fd-4954-a8ac-bac035891938"
}

resource "openstack_networking_router_interface_v2" "interface" {
  router_id = "${openstack_networking_router_v2.router.id}"
  subnet_id = "${openstack_networking_subnet_v2.subnet_private.id}"
}
```

As can be seen, it is necessary to add the ID of the external network to which the defined router will connect. Terraform does not allow to obtain this information by name, but only by ID. This must be obtained manually by logging into OpenStack.

5. The frontend instance. In this step the first instance is defined. To do it, the resource type "*openstack_compute_instance_v2*" is used. Within the resource, parameters such as the hostname, the selected instance ID with its type, the key pair already defined. The subnet dependency previously defined is also added and the network interface of the machine is created, which will have the IP defined within the *fixed_ip_v4* parameter, something mandatory for the frontend of the HPC cluster, since it will be used for resources such as NFS. An additional resource of type *openstack_compute_floatingip_associate_v2* is added in which the floating IP that is available is included and assigned to this instance that has just been defined, together with the IP of the interface that will be used to associate to this floating IP.
   Finally, it is important to highlight a parameter that has not been mentioned before, "*user_data*". This parameter is natively provided by the API and will define a script containing the commands to be executed on the machine once it is provisioned by Terraform, as highlighted in the definition of the infrastructure.

```
resource "openstack_compute_instance_v2" "terraform" {
  name            = "frontend"
  image_id        = "b39960c4-1f13-408f-8266-2fbf17079539"
  flavor_name     = "m1.tiny"
  key_pair        = "terraform"
  user_data       = "${file("install_frontend.sh")}"
  depends_on      = [openstack_networking_subnet_v2.subnet_private]

  network {
    name = "private_terraform"
    fixed_ip_v4 = "10.1.0.18"
  }
}
resource "openstack_compute_floatingip_associate_v2" "frontend_ip" {
  floating_ip = "193.144.42.201"
  instance_id = "${openstack_compute_instance_v2.terraform.id}"
  fixed_ip    =
"${openstack_compute_instance_v2.terraform.network.0.fixed_ip_v4}"
}
```

6. The compute instance. The definition of this resource is almost equivalent to the frontend one but without any specific IP. It is interesting to mention the "count" parameter, that will allow the administrator to define how many units of the defined instance are required in a simple way. In addition, in this case the *user_data* parameter includes another script. Both referenced scripts will be described in the next point.

```
resource "openstack_compute_instance_v2" "compute" {
  count           = "1"
  name            = "compute-${count.index + 1}"
  image_id        = "b39960c4-1f13-408f-8266-2fbf17079539"
  flavor_name     = "m1.tiny"
  user_data       = "${file("install_compute.sh")}"
  depends_on      = [openstack_networking_subnet_v2.subnet_private]
  network {
    name = "private_terraform"
  }
}
```

Once the configuration is finished, the deployment using Terraform environment can be launched in order to test the configurations. This deployment is launched by running the following commands:

```
$ cd terraform_openstack
$ terraform init
$ terraform validate
```

And finally:

```
$ terraform apply -auto-approve
```

The *-auto-approve* option avoid any prompt interaction, but it must be used carefully since it may result in changes without asking.

To destroy the infrastructure:

```
$ terraform destroy -auto-approve
```

When deployed, the HPC cluster can be used connecting to the instances using SSH.

5.1.3 Tests

In this section, each of the metrics and their aspects will be analyzed. Since the evaluation will be carried out according to the methodology defined in Chapter 4, section 4.3, only the particularities related to the use of Terraform that have been found in the process will be included. All the results will also be collected and summarized in the table also provided in section 4.3.

<u>Essential features:</u>

1. Performance

    a. Time of execution

The following command has been executed to evaluate this aspect, which, as indicated, performs the deployment of the infrastructure with Terraform. As indicated above, both commands have been executed five times.

```
$ time terraform apply -auto-approve
$ time terraform destroy -auto-approve
```

As a result, two values have been taken: the first and most important one, is the mean value of instantiation time. Additionally, the other value includes the post-configuration time, that is the time needed by the deployed cluster to apply its particular configuration.

32.184 sec. (instantiation) 13 mins (post-configuration)

b. CPU usage during deployment

The following command has been executed to evaluate this aspect, which, as indicated, performs the deployment of the infrastructure with Terraform.

```
$ while true; do top -a -pid $(pidof terraform | cut -d " " -f1) >>
cpulogfile; done
```

The result obtained is the following:

3%

c. Memory usage during deployment

The following command has been executed to evaluate this aspect, which, as indicated, performs the deployment of the infrastructure with Terraform.

```
$ /usr/bin/time -l terraform apply –auto-approve
```

The mean value that will be considered as the result is the following:

42 MB

d. API calls and network usage

As indicated in the analysis methodology. The network consumption during the execution will be analyzed using the Wireshark tool. After collecting the exchanged packets, the traffic that has as destination the IP address of the Cesga OpenStack server has been filtered. After that, the indicated filter was applied. The following values were obtained:

355 packets (0.16 MBytes)

2. Portability

As indicated in point 4.3, this point is evaluated taking into account the minimum Terraform requirements indicated by the provider. In this case, it is very important to note that no recommendations have been found for the open-source version of Terraform in relation to Memory and Storage, as they only exist for the Enterprise version. Since the two versions are actually different, it has not been possible to determine the exact value of the aspects to be evaluated.

However, it is important to note that no problems have been observed when executing the deployments.

a. Storage dependencies

Not found

b. Memory dependencies

Not found

c. OS dependencies

No dependencies

d. Third-party libraries required

No dependencies

3. Reliability

a. Log information

Available detailed logs setting up the TF_LOG (OFF, TRACE, DEBUG, INFO, WARN, ERROR to change the log level) and TF_LOG_PATH (to make it persistent) environment variables.

Regarding the callback log information of Terraform CLI commands give detailed information about what is running, even if they are executed in interactive or non-interactive mode.

```
Plan: 8 to add, 0 to change, 0 to destroy.
openstack_compute_keypair_v2.keypair: Creating...
openstack_networking_network_v2.network_private: Creating...
```

Full information.
Log levels available.

b. Number of failed launches

Throughout the cycle of tests and trials, no failed executions have been obtained as long as the deployment configuration has been complete and correct.

0 failed launches.

c. Workflow control

Terraform doesn't provide any control capability over the creation or deletion lifecycle of the infrastructure. This means that once the deploy or destroy command is issued, there is no possibility to modify this deployment, pause it or perform any other kind of action on it.

No workflow control

d. Status control

The status of the cloud provided used by Terraform is checked before starting the deployment. This information is saved to a file called *terraform.tfstate*, therefore it refresh any changes over resources. Even though, idempotency is broken if cloud provider suffers any issue or change during the deployment time.

Partial status control

4. Scalability and flexibility

a. Scalability capacity

After carrying out the deployment tests with 1, 2 or 3 computing nodes, it is observed that the environment instantiation time (total deployment time) does not vary. This is mainly due to the parallelization of the infrastructure creation. This is one of the main advantages of Terraform.

Fully scalable

b. Compatibility with cloud providers

The main cloud providers are natively supported. According to Terraform, it supports up to 100 different providers (cloud or not).

Fully supported

c. Capacity to integrate custom plugins

Due to the scarce documentation of the open-source version of Terraform, it is not entirely clear how to integrate third-party or proprietary plugins. However, the paid version does indicate instructions for integrating with third-party plugins. An example would be integration with Visual Studio Code for syntax checking.

Partially supported

d. Modularity

Modularity in Terraform exists through the "modules" component within the configuration of the infrastructures to be deployed. This allows the reuse of elements already instantiated in other deployments.

Fully supported

<u>Additional features:</u>

1. Cost

a. Open-source option

The open-source version exists and is fully supported, although limited to command-line use only.

Yes

b. Enterprise version option

There are two paid versions: an enterprise version and a cloud version. Regarding the open-source version, these versions offer a graphical interface that facilitates the use of the tool, better documentation, better integration with certain plugins and official vendor support.

Yes

2. Maintainability

a. Documentation

There is a powerful documentation regarding the configuration of the deployments and all the Terraform integrations with the different providers. However, there is a difference between the documentation available for the open-source version and that available for the paid versions. Additionally, it is observed that this tool has a lot of support from the community, since information of all kinds is provided within its Github, in addition to support.

Fully supported

b. Software releases policy

A new version of Terraform is released weekly. Each new version includes a "changelog" detailing new features added as well as bugs fixed. Additionally, it is worth noting that the upgrade process for the open-source version is by reinstallation only.

Fully supplied.
Manual updates

3. Security

a. Vulnerability control and tracking

Within the Terraform GitHub there is a specific section that includes the security policy followed by Terraform as well as the existing vulnerabilities that affect the product. Additionally, within the HashiCorp website [13] there is a section in which the vulnerabilities are described in great detail and follow-up is provided. As an example, the last published vulnerability is from May 24, 2022. This same procedure indicates how to fix it, as well as its implications and associated CVEs.

Yes

b. Identity management

There is no functionality to allow permission management within Terraform. In this way, the application trusts the permissions that users have at the operating system level to be able to use the tool or not.

Not supported

4. Complexity

a. High level language and syntax check

Terraform, as indicated above, uses its own language called HCL. Optionally, it is also compatible with JSON, but in practice it is not very supported since all the information from both the community and the manufacturer itself comes using HCL. Terraform itself has a tool to check the syntax of configuration files. When it finds errors, it indicates the reason why the error occurs and also the line it is on.

Propietary language.
Good syntax checker

b. Installation command

As has been indicated above, the installation process is very easy since it only consists in downloading the ZIP file that contains all the binaries, decompress it and execute.

Simple

The following table summarizes an evaluation of the different essential and additional features of the Terraform tool when deploying the reference HPC cluster.

Essential features		
Performance		
Aspect	Comments	Points
Time for creation	32.184 seconds (instantiation) 13 min (post-configuration)	2
CPU usage	3%	2
Memory usage	42 MB	2
Number of API calls	Transmitted 355 packages (0.16 MBytes)	2
Portability		
Aspect	Comments	Points
Storage dependencies	Not found	0
Memory dependencies	Not found	0
OS dependencies	No dependencies	2
Third-party libraries required	No dependencies	2
Reliability		
Aspect	Comments	Points
Log information	Full information. Log levels available	2
Number of failed launches	0 failed launches	2
Workflow control	No workflow control	0
Status control	Partial status control	1
Scalability and flexibility		
Aspect	Comments	Points
Scalability capacity: execution time for tests	Fully scalable	2
Compatibility with cloud providers	Fully supported	2
Capacity to integrate custom plugins	Partially supported	1
Modularity	Fully supported	2

Table 4 Terraform Essential features

Additional features		
Cost		
Aspect	Comments	Points
Open-source option	Yes	1
Enterprise version option	Yes	1
Maintainability		
Aspect	Comments	Points
Documentation	Fully supported	1
Software releases frecuency	Fully supplied. Manual updates	1
Security		
Aspect	Comments	Points
Vulnerability control and tracking	Yes	1
Identity management	Not supported	0
Complexity		
Aspect	Comments	Points

High level language and syntax check	Propietary language. Good syntax checker.	0
Installation command	Simple	1

Table 5 Terraform Additional features

Total number of points achieved by Terraform:

30 points out of 40

5.2 Ansible

5.2.1 Introduction to Ansible

Ansible is an open-source Infrastructure as Code (IaC) tool supported by Red Hat that allows automating all kinds of IT processes from service orchestration or application deployment, to configuration management or deployment of new infrastructures including those provisioned in the cloud. However, its modularity, flexibility and open-source nature make it possible to automate virtually any task carried out by an administrator. Using Ansible, a user can define specific **tasks**, within **playbooks**, to be performed automatically on a set of assets or in a defined environment. Tasks can be anything from preparing and deploying an entire infrastructure, to installing a specific version of software in the managed environment, to modifying a configuration, among the thousands of other options available.

The operation of Ansible is simple, and its main advantage and the reason for its simplicity is that it does not depend on an agent deployed on the assets to be managed. Ansible comprises two types of devices: control nodes and managed nodes. Control nodes are those where Ansible is installed and where the instructions to be executed on the managed nodes are defined. Usually there will be only one control node (although there can be backup nodes), while there can be as many managed nodes as necessary. Ansible connects to these managed nodes from the control node using an SSH connection (either via SSH keys or with username and password). Once connected, it deploys so-called "Ansible modules" to them, which are small pieces of software that allow user-defined configuration to be performed. Finally, these **modules** are executed, providing information to the control node about their status, to be ultimately deleted when they have fulfilled their task.

In relation to the structure of the configuration of deployments with Ansible, as indicated above, there are modules, which are used in tasks, which in turn are defined in **Ansible Playbooks**. These modules are generally integrated into Ansible, and are used within tasks, where they are provided with the appropriate parameters for their operation. A set of tasks define a Playbook, which would officially be defined as a set of configurations that allows the managed node to be brought to the desired state. The configuration of Ansible Playbooks is declarative, i.e., you define the state you want to reach rather than defining the steps to get there. These Playbooks are reusable and modular, written in YAML language. They are also defined as "idempotent", since they can be executed on a managed node as many times as desired, and will apply the appropriate configuration, with no consequences if the configuration has already been applied on the node.

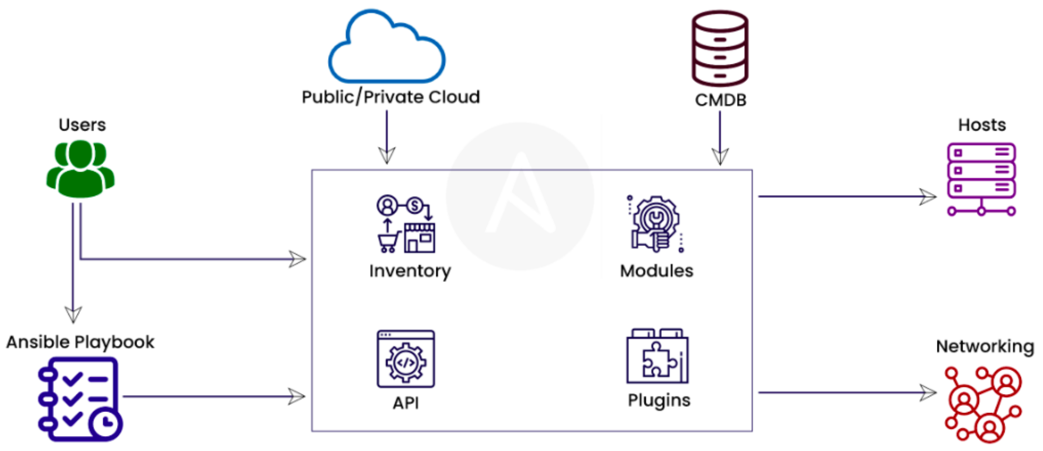


Figure 11 Ansible Architecture (© Veritis)

Ansible was initially designed to serve as a management tool for IT environments already deployed, but it has quickly adapted to new environments, such as the deployment and configuration of new infrastructures from scratch, as is the case in this thesis. Ansible has modules that allow it to integrate with the main public and private cloud providers, so that it is able to configure aspects such as the type of instance to deploy, including CPUs, storage, or network, and then the operating system, installed applications and their configurations using the CLI and REST APIs provided by different manufacturers natively.

All this makes Ansible a resilient, flexible and highly configurable IaC tool, which will allow to adapt to the automatic deployment of HPC clusters in Cesga's private cloud.

5.2.2 Configuration and cluster deployment

In a similar way as done before, the very first step is to install Ansible in the physical host. The simplest way to do it is through Python package manager (pip).

```
$ pip3 install ansible
```

This installation can be tested executing the following command:

```
$ ansible --version
```

Once Ansible has been installed on the host, it is mandatory to perform an additional step before entering the configuration file that will define the infrastructure of the HPC cluster to be deployed. Although Ansible does not need the cloud provider to be defined, is necessary to install the module to connect to the cloud provider, OpenStack in this case. This is done with the following command:

```
$ pip3 install openstacksdk
```

We found un bug [14] of the default version installed which was solved by downgrading the version of the module:

```
$ pip3 install openstacksdk==0.61.0
```

With this module installed, the next step is to proceed with the configuration of the defined HPC cluster. Again, a new directory will be created to hold all the specific Ansible configurations files to define the infrastructure. Inside, a new file will be created. This file will define the main Ansible Playbook:

```
$ mkdir ansible_openstack
$ touch ansible_openstack/main.yml
```

This file will contain the definition of all the elements that are needed in the HPC cluster to be deployed.

1. Information about the cloud provider. In the case of ansible, is not necessary to provide the information of the cloud provider to be used for the whole deployment. On the contrary, it is mandatory to specify the authentication for each resource, although basic authentication is enough. This will be seen in the below.

2. The SSH key pair. Again, this key pair will be used by Ansible to connect with the resources via SSH once they are deployed. OpenStack need the information of the public key, so the very first step is to upload it.

   At this point, where the first Ansible Task is defined, it is important to stop to talk about its structure, which follows the following guidelines:
   - Name: contains the name of the task. Subsequently it can be referenced from the Playbook to execute it atomically.
   - "*module_to_be_used*": the name of the module to be used is defined. In this case it is "*os_keypair*".
   - State: It is used to indicate if what is defined must exist in the infrastructure (present) or not (absent). Ansible will act to reach the defined point.
   - Auth: this section defines the authentication required to reach the resource, which is, in this case, OpenStack. It is mandatory to define the authentication URL, the user, the password, the name of the project that applies, its domain and the user's domain.
   - Finally, there are the definitions of the module used. In this case, the resource named "ansible" and the location of the public key are defined.

   Putting everything together, it looks like this:

```
  - name: Upload keypair
    os_keypair:
      state: present
      auth:
        auth_url: https://cloud.srv.cesga.es:5000/v3
        username: uscecmrr
        password: ***********
        project_name: hpc-project-uscecmrr
        project_domain_name: hpc
        user_domain_name: hpc
      name: ansible
      public_key_file: /Users/MariadalitRosales/.ssh/ansible.pub
```

3. The private network to connect all cluster instances. To deploy the private network with Ansible, is necessary to use two modules to create two resources: one of type "*os_network*" named "*private_ansible*", that defines the private network, and another of type *"os_subnet"* named "*subnet_ansible*", that defines the subnet inside the private network. This second resource uses the definition of the first to work, since the subnet needs a network. In this second resource the CIDR is also defined, 10.2.0.0/24 for the Ansible HPC cluster private network.
   It is important to note that both resources have the state "present" and in both cases, the authentication information is necessary.

```
- name: Create private network
  os_network:
    state: present
    auth:
      auth_url: https://cloud.srv.cesga.es:5000/v3
      username: uscecmrr
      password: ***********
      project_name: hpc-project-uscecmrr
      project_domain_name: hpc
      user_domain_name: hpc
    name: private_ansible

- name: Create private subnet
  os_subnet:
    state: present
    auth:
      auth_url: https://cloud.srv.cesga.es:5000/v3
      username: uscecmrr
      password: ***********
      project_name: hpc-project-uscecmrr
      project_domain_name: hpc
      user_domain_name: hpc
    network_name: private_ansible
    name: subnet_ansible
    cidr: 10.2.0.0/24
```

4. The router. Again, this router will redirect the traffic from the public network to the private network created above. The module used is "*os_router*", it is defined as "present" and its name will be *router_ansible.* The parameters that are needed are: the network to be connected, that is the public network available at OpenStack, and is named *public-network* is this case, and a list of the interfaces that need to be connected to this router. In this case, "*subnet_ansible*" is the subnet created above for the HPC cluster.

```
- name: Create router
  os_router:
    state: present
    auth:
      auth_url: https://cloud.srv.cesga.es:5000/v3
      username: uscecmrr
      password: ***********
      project_name: hpc-project-uscecmrr
      project_domain_name: hpc
      user_domain_name: hpc
    name: router_ansible
    network: public-default
    interfaces:
      - subnet_ansible
```

5. The frontend instance. In order to define the frontend instance of the HPC cluster, different modules are needed to configure all its parameters. The very first step is to define the fixed IP for the frontend inside the private subnet. This is done with the module "*os_port*", that will be named *frontend-port,* using the network *private_ansible* with the fixed IP 10.2.0.18. Once the fixed private IP for the instance is defined, the next step is to define the instance itself. To do this, the task named "Deploy frontend" with the module of type *"os_server"* is created. Inside this task, an instance named "*frontend"* is created, together with the image ID, the flavor, the key_name already created and finally the NICs for the instance, that is the resource named *frontend_port* that has been just created.
   Finally, in a similar way as done by Terraform, the parameter *userdata* provided by the API will contain the script with the executions that must be launched on the instance once it is deployed. This script was specified in the definition of the virtual HPC cluster to be tested and contains the necessary configuration for the cluster itself.

```
- name: Create port
  os_port:
    state: present
    auth:
      auth_url: https://cloud.srv.cesga.es:5000/v3
      username: uscecmrr
      password: ***********
      project_name: hpc-project-uscecmrr
      project_domain_name: hpc
      user_domain_name: hpc
    name: frontend-port
    network: private_ansible
    fixed_ips:
      - ip_address: 10.2.0.18
- name: Deploy frontend
  os_server:
    state: present
    auth:
      auth_url: https://cloud.srv.cesga.es:5000/v3
      username: uscecmrr
      password: ***********
      project_name: hpc-project-uscecmrr
      project_domain_name: hpc
      user_domain_name: hpc
    name: frontend
    image: b39960c4-1f13-408f-8266-2fbf17079539
    flavor: m1.tiny
    key_name: ansible
    userdata: "{{ lookup('file', 'install_frontend.sh') }}"
    nics:
      - port-name: frontend-port
```

   Also, it is important to note that for Ansible is not necessary to indicate the floating IP for the access to the instance, since it will automatically select it (something that can be modified with the parameter *auto_ip* that defaults to *yes* as defined below.

6. The compute instance. The definition of this resource is very similar to the one above. The main differences, apart from the name, are that no fixed IP is needed, the script deployed in the *userdata* field and the parameter *auto_ip* that indicates if a floating IP must be selected in order to have public access

from the Internet to the instance. In addition, unlike in Terraform, there is no simple way to deploy several instance with these equal characteristics, so the scalability becomes more complex since this task must be run in a loop to create multiple instances.

```
  - name: Deploy compute node
    os_server:
      state: present
      auth:
        auth_url: https://cloud.srv.cesga.es:5000/v3
        username: uscecmrr
        password: ***********
        project_name: hpc-project-uscecmrr
        project_domain_name: hpc
        user_domain_name: hpc
      name: compute-node
      image: b39960c4-1f13-408f-8266-2fbf17079539
      flavor: m1.tiny
      key_name: ansible
      network: private_ansible
      userdata: "{{ lookup('file', 'install_compute.sh') }}"
      auto_ip: no
```

Once the configuration is finished, the deployments of the defined virtual HPC cluster can be initiated by running the following commands. Firstly, it is recommended to check if there are any errors within the configuration:

```
$ cd ansible_openstack
$ ansible-playbook main.yml --syntax-check
```

And finaly, the deployment can be launched with the command:

```
$ ansible-playbook main.yml
```

When everything is deployed, the instances can be used by connecting to them using SSH.

It is important to note that there is no global functionality to rollback or destroy the deployed infrastructure. The only way is to modify the state of the elements of the current playbook from present to absent or create a new playbook that performs the reverse actions on the infrastructure. Another way is to use blocks in Ansible and define a rescue section to perform some tasks when some issue occurs. For this test, another Ansible Playbook has been created to reverse all the actions and destroy de environment created. This is done as indicated before but with the *state* option in each resource set to *absent*.

```
$ ansible-playbook main-reverse.yml
```

5.2.3 Tests

In this section, each of the metrics and their aspects will be analyzed in a similar way as it has been done in the section related to Terraform [5.1.3].

<u>Essential features:</u>

1. Performance

a. Time of execution

The following command has been executed to evaluate this aspect, which, as indicated, performs the deployment of the infrastructure with Ansible. As indicated previously, both commands have been executed five times.

```
$ time ansible-playbook main.yml
$ time ansible-playbook main_reverse.yml
```

As a result, two values have been taken: the first and most important one, is the mean value of instantiation time. Additionally, the other value includes the post-configuration time, that is the time needed by the deployed cluster to apply its particular configuration.

120.24 sec. (instantiation) 13 mins (post-configuration)

b. CPU usage during deployment

The following command has been executed to evaluate this aspect, which, as indicated, performs the deployment of the infrastructure with Ansible. It is important to note that the main process for Ansible is "Python", since Ansible runs over it.

```
$ while true; do top -a -pid $(pidof Python | cut -d " " -f1) >>
cpulogfile; done
```

The result obtained is the following:

16%

c. Memory usage during deployment

The following command has been executed to evaluate this aspect, which, as indicated, performs the deployment of the infrastructure with Terraform.

```
$ /usr/bin/time -l ansible-playbook main.yml
```

The mean value that will be considered as the result is the following:

42 MB

d. API calls and network usage

As indicated in the analysis methodology. The network consumption during the execution will be analyzed using the Wireshark tool. After collecting the exchanged packets, the traffic that has as destination the IP address of the Cesga OpenStack server has been filtered. After that, the indicated filter was applied. The following values were obtained:

657 packages (0.29 MBytes)

3. Portability

As indicated in point 4.3, this point is evaluated taking into account the minimum Ansible requirements indicated by Red Hat. In this case, Red Hat offers detailed recommendations to install and use Ansible in the most optimized way.

a. Storage dependencies

The vendor indicates that there is no specific requirement as to the type of disk to be used (it can be solid state or mechanical) and there is no specific requirement as to the size of the disk.

No dependencies

b. Memory dependencies

Red Hat recommends at least 4 GB to run Ansible in a proper way. This requirement is covered by most of the systems found nowadays.

Minimum dependency

c. OS dependencies

Supported natively in UNIX-based systems (either MacOS or Linux), including Windows Subsystem for Linux (WSL). Despite this, most of the data centers or systems designed for the installation of our HPC cluster or similar ones are based on Linux distributions.

No dependencies

d. Third-party libraries required

Yes, Python is required in order to be able to run Ansible. Additionally, there are some dependencies when executing deployments over cloud providers. In this case, the installation of the openstacksdk client is mandatory to support the test scenario defined for this Master's Thesis.

Yes

4. Reliability

a. Log information

There exists a log_path directive that administrators can include in the main ansible.cfg file to define the path for the logfile of this service. Also, the log level can be changed with the verbosity directive in the configuration file.

There are many callback plugins for different scenarios (however, most of them require a Linux host to work). Additionally, the callback information verbosity can be modified.

Full information.
Log levels available.

b. Number of failed launches

Throughout the cycle of tests and trials, some failed executions have been obtained when the deployment configuration was complete and correct. Most of these failures occurred due to timeout issues probably given by the high number of petitions done to the API of the cloud provider.

15% of failed launches.

c. Workflow control

Ansible provides with native options to perform workflow control. For example, there are specific tools that allow you to perform "rollback" of the operations performed. In addition, it allows Playbooks to be launched so that only specific tasks are performed.

Yes

d. Status control

Os_server_actions module [15] allows us to perform mainly actions over our precreated instances like (stop, start, rebuild). Moreover, the os_quota module [16] allows administrator to manage the quota on the cloud for any user. Ansible allow the administrator to get or set the status of the deployed resources in the cloud in real-time.

Yes

5. Scalability and flexibility

a. Scalability capacity

After carrying out the deployment tests with 1, 2 or 3 computing nodes, it is observed that the environment instantiation time (total deployment time) varies according to the number of compute nodes deployed. This is mainly due to the lack of parallelization while creating the infrastructure. However, if the administrator knows this issue, it can be overpassed by parallelizing manually the deployment of all the instances.

Partially scalable

b. Compatibility with cloud providers

The main cloud providers are natively supported. According to Terraform, it supports up to 100 different providers (cloud or not).

Fully supported

c. Capacity to integrate custom plugins

Ansible is a tool that is highly supported by several vendors. This makes a module available for virtually any need. Additionally, module development is officially supported and encouraged by Red Hat with official documentation and a repository.

Fully supported

d. Modularity

Ansible allows you to use your Playbooks in a completely flexible way. The tasks that make up each Playbook are completely reusable and you can even call other Playbooks from a task.

Fully supported

Additional features:

5. Cost

    a. Open-source option

The open-source version exists and is fully supported, although limited to command-line use only.

Yes

    b. Enterprise version option

There is an enterprise version. In comparison to the open-source version, this version offers a graphical interface that facilitates the use of the tool, and official vendor support.

Yes

6. Maintainability

    a. Documentation

There is extensive and easily accessible official documentation, including practical examples and additional resources. In addition, Ansible is a tool with great support from the open-source community.

Fully supported

    b. Software releases policy

Ansible community team releases a new minor version every 3 weeks and 2 major versions per year. Release roadmaps [17] are also available.

Additionally, the last version of Ansible (2.10) has divided the software into 2 packages (ansible and ansible-core). Therefore, to perform any upgrade it is recommended to uninstall the application and install the new version to avoid conflicts. This action is transparent for the playbooks according to the vendor.

Fully supplied.
Manual updates

7. Security

    a. Vulnerability control and tracking

Red Hat publishes the security issues on the customer portal once discovered [18]. Within this site each vulnerability is described and follow-up is provided. As an example, the last important published vulnerability is from December 2021 and includes the procedure to solve it as well as the CVEs associated [19].

Yes

b. Identity management

According to the official Ansible documentation, it supports identity management natively through RBAC. This provides an additional layer of security to that provided by the operating system itself where Ansible is deployed.

Supported

8. Complexity

a. High level language and syntax check

Ansible uses the standard YAML format, which makes it easy to describe the configuration due to the clarity of the format itself. This is also considered an advantage as it does not use its own language. Ansible also provides native tools to check configuration for syntax issues. When it finds errors, it indicates the reason why the error occurs and also the line it is on.

Not propietary language.
Good syntax checker

b. Installation command

Installation in the case of Ansible is somewhat more complex if Python is not previously installed on the computer, since this is an essential requirement. Once Python is installed, something that is very common since it comes pre-installed in most Linux distributions, the installation of Ansible is done using a command.

Simple

The following table summarizes an evaluation of the different essential and additional features of the Ansible tool when deploying the reference HPC cluster.

Essential features		
Performance		
Aspect	Comments	Points
Time for creation	120.24 seconds (instantiation) 13 min (post-configuration)	1
CPU usage	16%	1
Memory usage	42 MB	2
Number of API calls	657 packets (0.29 MBytes),	1
Portability		
Aspect	Comments	Points
Storage dependencies	No dependencies	2
Memory dependencies	Minimum dependency	1
OS dependencies	No dependencies	2
Third-party libraries required	Yes	1
Reliability		

Aspect	Comments	Points
Log information	Full information. Log levels available	2
Number of failed launches	15% of failed launches	1
Workflow control	Yes	2
Status control	Yes	2
Scalability and flexibility		
Aspect	Comments	Points
Scalability capacity: execution time for tests	Partially scalable	1
Compatibility with cloud providers	Fully supported	2
Capacity to integrate custom plugins	Fully supported	2
Modularity	Fully supported	2

Table 6 Ansible Essential features

Additional features		
Cost		
Aspect	Comments	Points
Open-source option	Yes	1
Enterprise version option	Yes	1
Maintainability		
Aspect	Comments	Points
Documentation	Fully supported	1
Software releases frecuency	Fully supplied. Manual updates	1
Security		
Aspect	Comments	Points
Vulnerability control and tracking	Yes	1
Identity management	Supported	1
Complexity		
Aspect	Comments	Points
High level language and syntax check	Not propietary language. Good syntax checker.	1
Installation command	Simple	1

Table 7 Ansible Additional features

Total number of points achieved by Ansible:

33 points out of 40

6. Conclusions and Future Lines

This chapter presents the answers to the essential objectives listed in section [1.2], the conclusion of the analysis carried out, the contribution of this Master's Thesis and the lines of future work.

6.1 Achievement of key objectives and Conclusions

The main objectives of this work have been as follows:

- Identify whether it makes sense to deploy high-performance systems, i.e. HPC clusters, in private, public or hybrid cloud environments.

In the Introduction and Motivation sections, the reader has been shown the increase of cloud environments in the HPC market in recent years. This increase is remarkable not only because it represents a large subset of the entire HPC market but also because of the speed of its growth, demonstrating that it is a booming technology. There are several success stories of users making use of HPC resources in public cloud environments, although so far the investment in private environments is higher than in public clouds.

- Whether there is a reliable method to help evaluate infrastructure management tools for automated deployment of HPC clusters in cloud environments.

Nowadays, there is no standardized method for comparing automation tools for the deployment of high-performance systems in cloud environments. However, there are tools of different types that claim to have the ability to help deploy an HPC cluster automatically, either generically (IaC tools) or specifically (Over IaC tools). Given the community's lack of knowledge for such a specific application of infrastructure deployment automatism, it was necessary to conduct an experiment with tools developed specifically to deploy an HPC cluster in the cloud automatically. Although the conclusions of this experiment will be described below, conducting it was necessary to have the ability to steer the analysis in the right direction. Therefore, despite the lack of information on the subject, in order to solve this problem, and thanks to our own experience together with the reading of multiple papers about similar comparisons, in Chapter 4 we propose an evaluation method that takes into consideration multiple needs of an HPC environment. First of all, a specific environment is defined on which the tests will be performed, and then the tools to be analyzed are chosen from among the most popular on the market. The core of this chapter defines the exact methodology of the analysis, presenting the metrics to be considered and dividing them into essential and additional ones. These metrics, in turn, are divided into aspects that make them up, something that allows an evaluation to be carried out in an atomic and simple way. Since an analysis lies not only in the metrics, but also in the way they are measured, this chapter also describes the technical and non-technical resources needed to correctly measure each of the aspects that make up the metrics. Finally, a specific HPC environment is defined that may be of great interest for automation and in which the reader will be interested. Therefore, it can be concluded that Chapter 4 fulfills this objective, since it manages to define a reliable and effective evaluation method to compare tools that allow the automation of HPC clusters in the Cloud.

- If there is any leading tool in the market that stands out from the rest in terms of how feasible its implementation, adoption and development is, to manage these environments, based on the method of the first question.

As indicated above, despite the fact that the objective sought is very specific, there are different tools that more or less specifically make it possible to achieve it. Obviously, the most logical thing to do was to start by analyzing those tools that claimed to provide specific capabilities for not only the deployment, but also the management of HPC clusters in the Cloud automatically. For this reason, and in view of the complete lack of knowledge about these tools, Chapter 3 analyzes two of the best known tools for this purpose: IM and ElastiCluster. The conclusion after the various tests carried out, as can be read in the same chapter, is that although these tools have great potential, they are in a very low state of maturity. For this reason, pushed furthermore by the lack of support from the community, the scarcity of their documentation and therefore, the difficulty of their customization and adaptation to different types of deployments, it was mandatory to discard them in favor of tools that although they do not promise specific functions for HPC environments, they do have enough flexibility to adapt to the needs defined in the analysis.

Thus, Chapter 5 compares the two most popular, and also most powerful, IaC tools: Terraform and Ansible. As can be read in this chapter, and taking into account the analysis methodology applied, both tools are able to perform the deployment without any kind of inconvenience, and manage to run the prototype cluster efficiently and without major problems. However, it is necessary to talk about several particularities, always taking into account the metrics followed:

In relation to **performance**, Terraform's efficiency when deploying stands out. This tool manages to build the environment in less time than Ansible, and with a lower consumption of resources, especially CPU and network.

Portability, despite being very even in practice, is in practice surpassed by Ansible due to its higher quality documentation. It is noteworthy that Terraform manages to run without any dependencies, while Ansible requires Python, something minor, since it is preinstalled in almost all Linux distributions.

In terms of **reliability**, each has its pros and cons, with Ansible being the clear winner, and although it has suffered several errors when deploying the environment (something that is caused by its high network demand), it provides native tools that will give the administrator confidence about what is being executed and in what way, allowing additional actions to the deployment.

Scalability, although on paper it seems to be on Terraform's side, as it is natively capable of scaling without major time consumption, ends up going to Ansible due, again, to its flexibility. Ansible will allow scaling in parallel if its tasks are scheduled in a specific way and furthermore, it supports great compatibility with vendors, plugins and additional modules that will ultimately make the job easier for administrators.

Regarding additional features, we find an identical **cost** in both tools, since they provide an open-source version more than sufficient for the vast majority of needs and easy **maintenance** since the vendors provide adequate information in all cases. In terms of **security**, we find that Ansible provides additional security measures, such as an RBAC-based control for the execution of Playbooks, which can reduce the risk of unwanted executions. Finally, the **complexity** of both tools is low, although the simplicity of Ansible is considered superior since it uses a generic language, rather than a proprietary language as used by Terraform.

With all of the above, we find that **Ansible is positioned as the more recommendable** tool, but nearly followed by Terraform. Anyway its flexibility, resilience and ability to adapt to any environment makes it a very powerful and capable tool. Additionally, due to the immense amount of modules available, many of which can already be found in Playbooks with defined objectives, make it the right tool not only to perform the deployment, but also to manage the deployed infrastructure once it is in production. This makes it not surprising that one of the tools evaluated in the experiment, ElastiCluster, uses it below to manage deployments and configurations, which makes it important to follow the evolution of this last mentioned tool.

Terraform, on the other hand, is more of a tool limited to the deployment and destruction of virtual environments, but it is undeniable that it performs this task in a very simple and efficient way, although without providing any highlighted additional capabilities.

6.2 Contribution

Currently, multiple comparisons can be found between Ansible and Terraform as they are the most used and supported tools of this type by the scientific community. However, there is no detailed document that evaluates the use of these tools in cloud environments for the creation of an HPC cluster.

This Master's Thesis presents a starting point in the feasibility study of the installation of HPC environment automation tools in cloud platforms, providing practical results on the installation, deployment, and impact on the daily tasks of the administrators of these environments. In addition, the selected methodology and metrics serve as a starting criterion for the establishment of a first standard in the development of high-performance computing environments from a cloud-agnostic perspective.

6.3 Future Lines

Self-service capabilities are essential for cloud management platforms to be flexible and adaptable to frequent changes in the technology sector, without being an obstacle or an almost impossible challenge for administrators.

Over IaC tools have been shown to offer in their theoretical basis all the advantages to make the deployment and development of these booming environments as simple as possible. However, their premature state means that their adoption is not yet sufficiently attractive to those responsible for IT environments. Once these tools are at a more mature stage, it would be appropriate to perform a further analysis on them to assess the point at which the inclusion of these tools is better than the deployment of underlying tools such as Ansible or Terraform.

On the other hand, one of the main advantages of cloud environments is the elasticity since it can increase and decrease the number of instances automatically depending on the user load. This feature could not be evaluated since the prototype cluster was deployed statically indicating the number of instances at startup. It is recommended to evaluate this auto-scalability and dynamism feature using the corresponding clustering APIs.

A comparison between declarative and imperative tools would also be of interest since currently no robust methodology has been developed to evaluate tools with different approaches.

Finally, it has been shown that most IaC tools are offered as command line tools, while Over IaC tools offer the advantage of a more user-friendly visual environment for the administrator and users, although often restricted to a proprietary syntax or a specific cloud. The development of an easy-to-install web tool that includes different panels for the user and administrator, that allows different authentication models in a simple way, that includes reporting and monitoring, along with a really simple model that allows the administrator to set his favorite tool for cluster deployment, would be of great interest to the community as this tool does not currently exist in free and cloud-agnostic mode and would be a turning point in the administration of these complex environments.

Bibliography

[1] "Informática de alto rendimiento (HPC) — Microsoft Azure." (), [Online]. Available: https://azure.microsoft.com/es-es/solutions/high-performance-computing/#industries.

[2] R. Aljamal, A. El-Mousa and F. Jubair, "A comparative review of high-performance computing major cloud service providers," *2018 9th International Conference on Information and Communication Systems (ICICS)*, 2018, pp. 181-186, doi: 10.1109/IACS.2018.8355463
[Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8355463.

[3] "European weather cloud." (), [Online]. Available: https://www.europeanweather.cloud/about.

[4] H. A. Imran, S. Wazir, A. J. Ikram, A. A. Ikram, H. Ullah and M. Ehsan, "HPC as a Service: A naïve model," *2019 8th International Conference on Information and Communication Technologies (ICICT)*, 2019, pp. 174-179, doi: 10.1109/ICICT47744.2019.9001912
[Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9001912.

[5] A. G. Carlyle, S. L. Harrell and P. M. Smith, "Cost-Effective HPC: The Community or the Cloud?," *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, 2010, pp. 169-176, doi: 10.1109/CloudCom.2010.115
[Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5708448.

[6] A. Gupta and D. Milojicic, "Evaluation of HPC Applications on Cloud," *2011 Sixth Open Cirrus Summit*, 2011, pp. 22-26, doi: 10.1109/OCS.2011.10
[Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6200551.

[7] D. Vladusic and D. Radolovic, "Infrastructure as Code for Heterogeneous Computing," *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2020, pp. 1-2, doi: 10.1109/SYNASC51798.2020.00011
[Online]. Available: https://ieeexplore.ieee.org/document/9357104.

[8] "TOSCA simple profile in YAML version 1.0." (), [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html.

[9] " *Establishing Qualitative Software Metrics in Department of the Navy Programs*
"[Online]. Available: https://www.readcube.com/articles/10.21236%2Fada622682.

[10] "Provider issue (openstack), VM boot up error "network XXX requires a subnet in order to boot instances on" · issue #775 · terraform-provideropenstack/terraform-provider-openstack," GitHub. (), [Online]. Available: https://github.com/terraform-provider-openstack/terraform-provider-openstack/issues/775.

[11] Terraform, original-date: 2014-03-13T22:25:48Z, Jul. 25, 2022. [Online]. Available: https://github.com/hashicorp/terraform.

[12] "Issues · hashicorp/terraform," GitHub. (), [Online]. Available: https://github.com/hashicorp/terraform/issues.

[13] "Latest security topics," HashiCorp Discuss. (), [Online]. Available: https://discuss.hashicorp.com/c/security/52.

[14] "Bug #1975497 "openstacksdk 0.99.0 failure in compute.v2.server i..." : Bugs : Openstack i18n," Launchpad. (), [Online]. Available: https://bugs.launchpad.net/openstack-i18n/+bug/1975497.

[15] "Os server actions - perform actions on compute instances from OpenStack — ansible documentation." (), [Online]. Available: https://docs.ansible.com/ansible/2.3/os_server_actions_module.html.

[16] "Os quota - manage OpenStack quotas — ansible documentation." (), [Online]. Available: https://docs.ansible.com/ansible/2.3/os_quota_module.html.

[17] "Ansible roadmap — ansible documentation." (), [Online]. Available: https://docs.ansible.com/ansible/latest/roadmap/ansible_roadmap_index.html#ansible-roadmaps.

[18] "Red hat CVE database - red hat customer portal." (), [Online].Available: https://access.redhat.com/security/security-updates/#/cve?q=ansible&p=1&sort=cve_publicDate%20desc&rows=10&documentKind=Cve.

[19] "CVE-2021-20180- red hat customer portal." (), [Online]. Available: https://access.redhat.com/security/cve/cve-2021-20180.

[20] E. Joseph et al., "The hyperion research team," p. 92, 2022 [Online]. Available: https://hyperionresearch.com/wp-content/uploads/2022/06/Hyperion-Research_ISC22-Market-Update_May-30-2022_Combined.pdf.

[21] "Assess resource consumption with ansible callback plugins — enable sysadmin." (), [Online]. Available: https://www.redhat.com/sysadmin/ansible-callback-plugins-metrics.

[22] "Callback plugins — ansible documentation." (), [Online]. Available: https://docs.ansible.com/ansible/2.8/plugins/callback.html.

[23] "Browse providers — terraform registry." (), [Online]. Available: https://registry.terraform.io/browse/providers.

[24] GitHub HashiCorp. "HashiCorp," HashiCorp. (), [Online]. Available: https://github.com/hashicorp/terraform/releases.

[25] HashiCorp. "HashiCorp," HashiCorp. (), [Online]. Available: https://www.hashicorp.com/security.

[26] "Latest security topics," HashiCorp Discuss. (), [Online]. Available: https://discuss.hashicorp.com/c/security/52.