Adaptive Event Driven Infrastructure

João Marcos Gaspar Campos

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor(s): Prof. Prof. Fernando Henrique Côrte-Real Mira da Silva
Eng. Simão Pedro Patrício da Silva

Examination Committee

Chairperson: Prof. Luís Manuel Antunes Veiga
Supervisor: Prof. Fernando Henrique Côrte-Real Mira da Silva
Member of the Committee: Prof. João Nuno De Oliveira e Silva

October 2021

Acknowledgments

I would like to express my gratitude to Prof. Fernando Mira da Silva for is guidance throughout this master thesis. His insights, support and suggestions were essential in the making of this document.

I would also like to thank Eng. Simão Silva for his great help in making the implementation possible. For his availability and direction while assuring all the required technologies and systems were accessible for use.

Resumo

Sistemas baseados em computação em nuvem apresentam vários desafios em termos de partilha de recursos, alocação e administração.

Com a introdução do conceito de virtualização e dos centros de dados de larga escala, operações tais como criação e destruição de recursos virtuais têm sido alvo de especial atenção no que toca à performance e eficiência. Automação de tarefas de administração e configuração dinâmica são alvos principais no tentativa de providenciar integração célere, testagem, e distribuição de software e infraestrutura.

A crescente complexidade e escala de sistemas torna claro que são necessárias melhores ferramentas de apoio para atingir formas de configuração e gestão ideais. Este suporte de decisões pode utilizar indicadores comportamentais de sistemas em operação aplicando métodos de otimização e algoritmos de aprendizagem automática.

Esta dissertação introduz uma arquitetura para integração de vários sistemas que possibilita a recolha e processamento de dados que facilitam a gestão autónoma de infraestruturas de computação em nuvem e descreve a estratégia adoptada para a implementação e testagem do modelo proposto.

Palavras-chave: Computação em Nuvem, Infrastrutura, Sistemas Adaptativos, Logs, Métricas, Eventos

Abstract

Cloud computer systems have several challenges in the domains of resource sharing, allocation and administration.

With the advent of very large data centers and virtualization, operations such as creation and deletion of virtual resources have been subject of special attention concerning performance and efficiency. Automation of administrative tasks and dynamic configuration are primary goals in an effort to provide fast integration, testing and deployment of software and infrastructure.

The increase in complexity and scale of large systems has made clear that more help is needed for optimal configuration and management. This support can come from leveraging behavioral indicators from running systems with the application of optimization methods and machine learning algorithms.

This thesis introduces an architecture for integration of several systems for the collection and processing of data to facilitate the automated management of cloud infrastructures and describes the adopted strategy for implementation and test of the proposed model.

Keywords: Cloud Computing, Infrastructure, Adaptive Systems, Logs, Metrics, Events

Contents

List of Tables

List of Figures

Acronyms

API Application Programming Interface. 11, 19, 20, 30, *Glossary:* API

CP Control Program. 3

CPU Central Processing Unit. 20, 22, *Glossary:* central processing unit

ESB Enterprise Service Bus. 14

IaC Infrastructure as Code. 11, 12

IT Information Technology. 1, 9, 12, 16, 17, *Glossary:* IT

json JavaScript Object Notation. 29, *Glossary:* json

OS Operating System. 4–6, *Glossary:* operating system

PCI Peripheral Component Interconnect. 5, *Glossary:* PCI

SAN Storage Area Network. 9

SDN Software-defined Network. 8

SDS Software-defined Storage. 9

SOA Service-Oriented Architecture. 14, 15, 22

SSH Secure Shell. 14, 26, *Glossary:* ssh

VLAN Virtual Local Area Network. 7

VM Virtual Machine. 3, 20, 25

VMM Virtual Machine Monitor. 3, 4

VPN Virtual Private Network. 7

Glossary

API Application Programming Interface. A set of specifications that a service of software defines to communication by a third party. 11

hostname A distinguishable name or label assigned to a computer or server. 12

IT Information Tecnhology is the field of study or use of systems for creating, storing, sharing and processing of electonic information. 1

json Javascript Object Notation is a data representation format designed for easy human read and write operations as well as easy parsing. 29

kernel The core module of an operating system. It contains the basic functions of the operating system and manages the interactions with the computer hardware. 5

mainframe A computer or set of computers used by organizations for large volumes of processing work. They are charecterized by their size, reliability, and high capability for computing and storage. 3, 9

markup language A stardart for text documents. A set of anotations or tags inserted in a text document that define a structure format for the content. 14

monolithic application An Application that combines the interface, data access, code and logic in a single self contained program. It is usually independent from any other applications. 14

open-source Open-source indicates a software where source code is freely available. It is generally licensed without restrictions for distribution or modification. 6

operating system A low-level software that controls a computer hardware and manages the processing of multiple programs or applications. 4

PCI Peripheral Component Interconnect. A standard for an interface bus that is commonly used to connect hardware components in a computer. 5

ssh Secure Shell. A communication protocol that enables encrypted computer-to-computer connections. Among other uses it allows remote terminal login or remote command execution over an unsecured network. 14

Chapter 1

Introduction

Since the beginning of computer science, resource sharing and allocation of hardware infrastructures have been a field of interest, research, and development. It had an impact in a wide range of computer science areas from the development of hardware and software all the way to management practices and governance policies in Information Technology (IT).

There are a variety of approaches and practices for architecture, management, and administration of shared infrastructure. Implementation choices are not solely based on capabilities. Sometimes cost, ease of integration, and versatility have a higher than expected weight in the selection of technologies. This requires management systems composed of large and diversified layers and modules.

1.1 Motivation and Goals

The tasks of configuration and management of resources in large datacenters, is a complex one. Some of the primary goals are to minimize failures or maximize performance with the least possible amount of risk. This is especially difficult when there are a large number of systems composed of many modules.

The behavior of the system is key for the collection of relevant insight. Metrics, events, and logs are generated in large quantities. Exploration of this data may help in making administration decisions.

The goal of this thesis is to contribute to the dynamic management and automatic configuration efforts in a datacenter environment.

1.2 Contributions

This thesis contributes to the challenges present in datacenter operations by proposing an architecture for automated infrastructure management. The suggested design includes a platform for applying optimizations techniques, a way of integrating multiple sources of data, and the possibility of making changes to existing infrastructure.

It is also presented a working proof-of-concept of such platform, where a service is changed and modified in an automated manner.

1.3 Structure of this Document

In chapter **2 Background**, it is provided a summary about virtualization and datacenter management. The objective is to understand the development that led to current technologies and practices in datacenter environments. Also a brief overview over the tools and methodologies used in infrastructure management.

Chapter **3 Related Work** presents some examples of current works that make use of innovative techniques to provide administrative support.

Chapter **4 Solution Architecture** proposes a model for dynamic infrastructure modification that integrates existing platforms and services. A real case proof-of concept is detailed in chapter **5 Solution Implementation**.

Chapter **6 Evaluation and Results** covers the tests and evaluation done to the implementation.

Finally chapter **7 Conclusions** wraps up this document aligning future work.

Chapter 2

Background

This section presents relevant developments that impact current practices in the infrastructure and operation of information systems and datacenters.

It starts by presenting an overview of the concept of virtualization, namely its inception and how it was applied to computing, networks, and storage. It analyses some technologies and methodologies used in datacenters. It also introduces the notion of cloud and how it affects the way in which infrastructure is used and operated.

Finally it explores the concept of adaptive systems and the application of decision support systems to the administration of infrastructure.

2.1 Computer Virtualization

The application of the virtualization concept in computer systems can be traced to the first mainframes in the 1960s. Even though the definition has evolved since its inception, virtualization was developed as a way to logically divide physical resources [1].

The Virtual Machine Facility/370 IBM [2], created in 1972, allowed multiple pieces of code to make use of the same machine resources independently. While the virtualization concept appeared in other works before, this project marks an important milestone. It had a control program (CP) that created virtual versions of the hardware.

In 1974, Popek *et al.* [3], reinforced the idea that a virtual machine (VM) must be an efficient and isolated duplicate of the real machine. They proposed the name virtual machine monitor (VMM) for the program used to create this environment. A VMM must have three fundamental properties. *Efficiency*, *Resource Control*, and *Equivalence*.

Resource Control The VMM must control the system resources. To accomplish this, under certain circumstances, intercepts explicit calls of the guest programs.

Equivalence The effect of any program running on the VMM must be identical to the case where the program runs on the original hardware uncontrolled.

Efficiency A large subset of the program instructions must be executed directly in the real processor with no control by the VMM.

The notion of virtual machine is enhanced with these properties. Previous concepts like time-sharing and virtual memory become only an aspect of the whole model. It also creates a distinction between the VMM and simulators where all instruction are interpreted [3].

In the 90's cost, availability, and compatibility made the x86 architecture family of processors popular and widespread. Virtualization was possible, however the hardware implementation lacked support. A software-based approach required all code to be interpreted, leading to low efficiency levels. To meet all of the VMM properties the proposed solution was to perform binary translation at runtime [4].

Eventually hardware assistance for virtualization was extended for the x86 architecture. Some of these implementations are known as VT-x from Intel and AMD-V from AMD, two of the largest semiconductor chip manufacturers [5].

2.1.1 Hypervisors

Virtual machine monitors continued to evolve and other terms emerged such as hypervisor. It probably derived from the idea of a supervisor of supervisors. Supervisor is the name sometimes used to refer to an operating system (OS) [6].

Hypervisors can be separated into two types, *Native* (Type I) and *Hosted* (Type II) [7]. A Type I hypervisor runs directly on top of the hardware. It has a single purpose of scheduling and allocating hardware resources to virtual machines. A Type II hypervisor hypervisor is developed as an application to run on top of an existing operating system. Figure 2.1 shows a visual model that helps to better understand the differences.

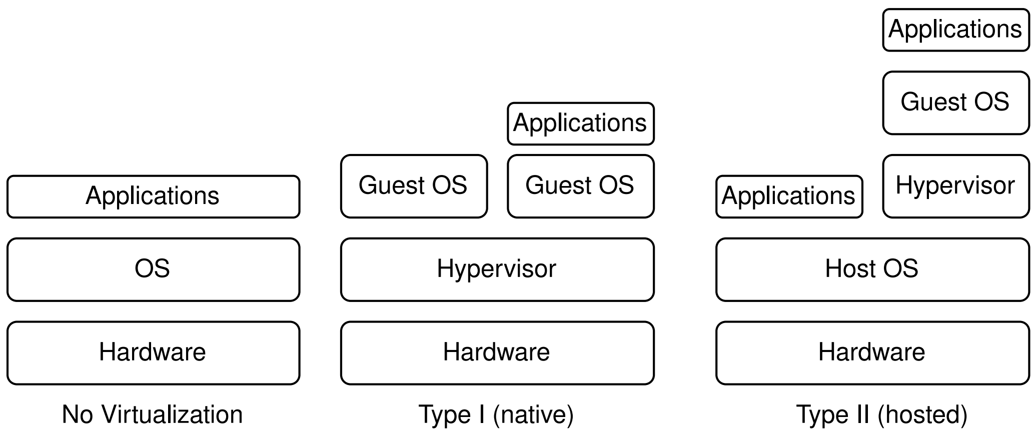


Figure 2.1: Virtualization models

There are several examples of Type II (Hosted) virtualization like QEMU [8], VirtualBox [9], VMware Workstation [10]. As for Type I (Native) well recognized options are VMware ESXi [11], Microsoft Hyper-V [12], and Xen [13].

There are also hypervisors designed as extensions for the OS kernel. Examples are Kernel-based Virtual Machine (KVM) for linux kernel since 2007 [14] and Bhyve hypervisor for FREEBSD [15]. Even though they are part of the kernel and may be seen as a native hypervisor, there are still the normal functions of an operating system present. Table 2.1 shows a comparison of some hypervisor features. It includes as features the notion of *Live migration* and *PCI Passthrough*. The first is a way of changing the host without stopping the guest operating system [16]. The advantage is that from the guest perspective there is no downtime. PCI Passthrough enables the guest to directly access PCI devices such as networks or graphical cards [17]. The performance of the guest increases largely. KVM and Xen have fewer limitation in respect to host resources such as CPU and RAM.

Table 2.1: Comparison of Hypervisor Features

	ESIx	Hyper-V	KVM	Xen
Version	6.7	Server 2019	RHV4.x+KVM	7.1
Current Developers	VMWare	Microsoft	Red Hat	Linux Foundation
License	Proprietary	Proprietary	LGPL	GPLv2
Type	native	native	hosted	native
Host CPU	x86, x86-64	x86-64	x86, x86-64, ARM	x86, x86-64, ARM
Live Migration	Yes	Yes	Yes	Yes
PCI Passthrough	Yes	Yes	Yes	Yes
		Host		
Max. Logical CPUs	768	320	768	288
Max. RAM	16 TB	4 TB	12 TB	5 TB
Max. Virt CPU/Host	4096	2048	4096	128
		Guest VM		
Max. CPU (Logical)	256	240	384	32
Max. RAM	6 TB	12 GB	6 TB	1.5 TB
		Cluster		
Max. Nodes	64	64	32	64
Max. VM	8000	8000	4000	N/A

[18–23]

A performance comparison of these hypervisors is made in [18]. The experiment put guest operative systems performing a database workload. Then, some performance metrics are measured. The results of CPU usage, Memory access and Disk I/O operations, do not show significant large variations overall. In this specific case Hyper-v showed slightly better CPU performance, KVM slightly better memory efficiency and ESXi better performance in disk I/0 (input/output operations).

Concerning features and performance, these technologies are similar. The fact that KVM and XEN are free and open-source software may influence a choice in their favor. The cost of licensing ESIx or Hyper-V will grow along with host nodes added. Other aspects are also important to consider when making a choice: ease of implementation, compatibility with other specific projects, or development environment requirements.

2.1.2 Containers

Container based Operating System is another approach to virtualization. It is also called *Operating System-level virtualization*. As indicated in the name, isolation between applications is achieved by instantiating a copy of the operating system resources required by each application. They are often defined as lightweight virtual machines.

The model presented in Figure 2.2 helps in understanding the container concept. The host operating system supports the container manager. In the container model, binaries and libraries are not shared among applications.

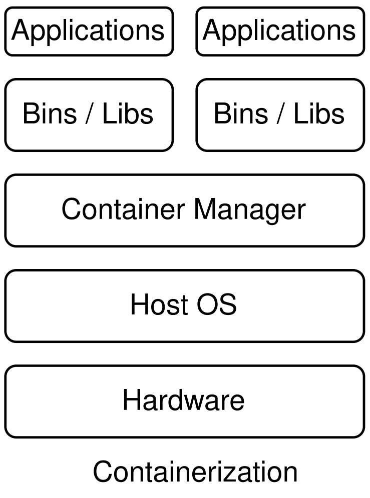


Figure 2.2: Containerization model

When the model was proposed, it aimed at improved performance and scalability [24]. In [25] it is presented an early comparison of container technologies against a Xen hypervisor in a high performance computing scenario. It showed that they offered near-native performance of CPU, memory, disk and network.

There are however concerns in security. Containers offer a lower level of isolation than hypervisors. The operating system exposes a large collection of system calls that broaden security risks. This occurrence is well known. Security in containers is an area that is still the object of study and developments

[26, 27].

2.1.3 Unikernels

Some computational tasks can be highly specialized. Sometimes they are performed by a single process application. In these cases, it is only used a reduced set of libraries from the operating system. These can be extracted along with strictly required binaries and the desired application. The result is a lightweight bootable virtual machine [28]. This is the idea behind the unikernel.

Figure 2.3 shows the model alongside the previous approaches of containers and hypervisors.

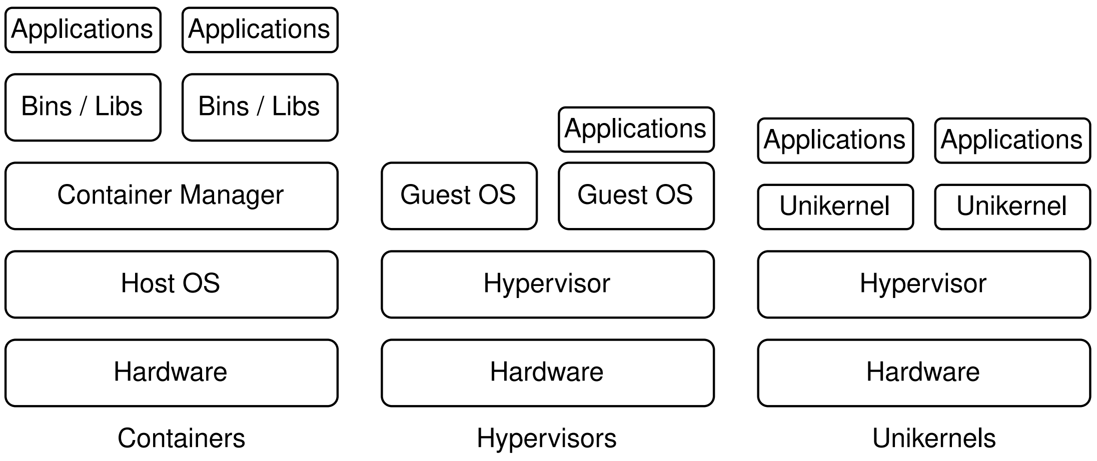


Figure 2.3: Unikernel and other models comparison

Unikernels still run on a hypervisor. One advantage is the level of isolation. They don't share the same operating system like containers but are still lightweight.

In [29] it is presented a comparison of creation times between a container, a linux virtual machine, and a unikernel. It can be observed that unikernels have very similar results to containers while still booting a virtual machine. As for the linux virtual machine, it takes around 3 times more time and suffers greatly with the increase of concurrent instances in the same hypervisor.

2.2 Network Virtualization

Stability, scale, and management are some of the difficulties of a classical network architecture. One of the first problems arising with virtualization is the requirement of coexistence of several networks in the same infrastructure. Virtual Local Area Networks (VLANs) are one solution [30]. They join devices in the same broadcast domain regardless of the device they are physically connected to.

Overlay Networks are networks built upon an existing network. They can accomplish this by using tunneling and encapsulation. This technique permits the deployment of new features and fixes to specific problems without added cost [31].

Virtual Private Networks (VPNs) are another important example. They can connect multiple sites with secure tunnels over shared and public networks [31].

In traditional network hardware, like switches or routers, one can identify three planes of operation. *Data*, *Control*, and *Management*. Reception and transmission of packets takes place in the *Data* plane. Its responsibilities are packet buffering, scheduling, header modification, and forwarding. The *Control* plane registers forwarding rules. Here the network topology can be effectively changed. As for the *Management* plane, it monitors the switch and is used to configure the other planes. By separating the *Control* and *Data* planes, and having a centralized controller it is possible to create a new type of network management, which may contribute to more flexible virtualization solutions.

One application of this concept are Software-defined Networks (SDNs). It uses a set of protocols and compatible hardware. Devices are somewhat agnostic to network topology or general security policy. They just reflect the direct rules ordered by the controller in their *Data* plane [32]. In Figure 2.4 it can observed how the SDN model changes network architecture and security.

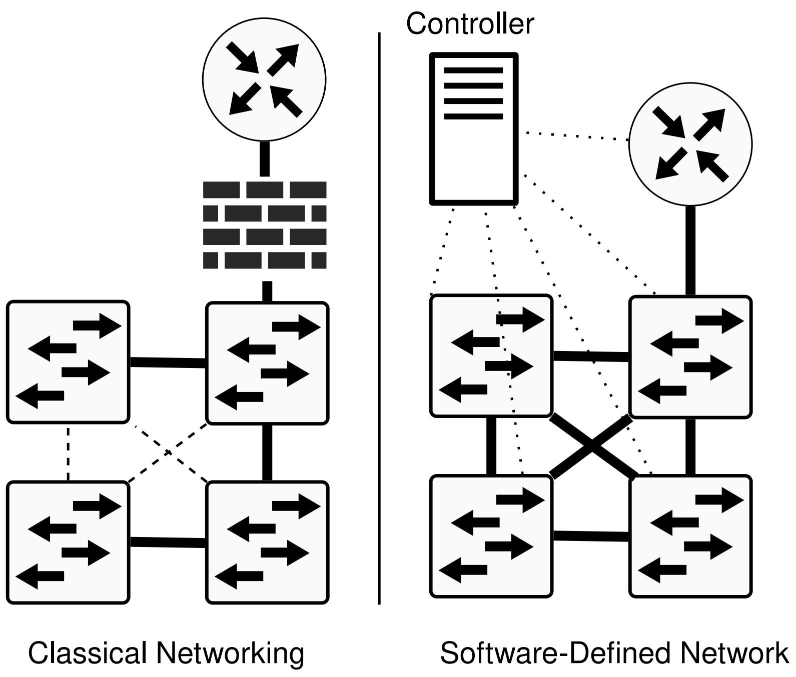


Figure 2.4: Network Architecture Models

OpenFlow was proposed in 2008 as a protocol for the communication between the controller and the network devices [33]. Nowadays SDNs are more associated to a central controller, regardless of the communication protocol used. FloodLight, OpenDaylight and Beacon are examples of software that implement controller platforms [34].

SDNs are an interesting solution for dynamic environments. For example in datacenters they simplify network creation and deletion. There are some aspects that need further development. For instance the architecture of controllers in large-scale environments or the possible memory limitations of network devices [35, 36].

2.3 Storage Virtualization

In the first computer mainframes data was stored in specialized devices, usually tapes or disks.

Eventually computer servers evolved to have storage internally. This *server-centric* architecture had some disadvantages. Most server have a limited number of slots for storage devices. Maintenance tasks on one server can make its data unavailable. To tackle this problem an *information-centric* approach was developed. Storage devices are separated from the server. They can be shared among a large number of computers increasing availability and capacity [37].

Storage Area Networks (SANs) are an implementation of a centralized data storage at the *Block level*. They provide optimizations like dynamic growth and shrinking of virtual disks. By striping data across different disks they can also provide faster access times. SANs also include the high speed network system and protocols that connect the computing servers to storage servers [38, 39].

The idea of separating the control and data planes, originally proposed for SDN architectures, yields several benefits when applied to data storage. As the data creation grows exponentially in a datacenter environment, scalability without loss in efficiency is a must. Software-defined Storage (SDS) aims to ease data configuration management by breaking the vertical alignment of conventional storage design [40].

When a virtual machine triggers a request to a virtual storage device, several things happen, including requests to the hypervisor, packet routing, and input output operations in a storage server. This complex path makes it hard to enforce widespread policies. For instance, it is difficult to differentiate requests based on their contents in each step.

IOFlow was proposed as a storage architecture with a central controller that guarantees performance and security when accessing data resources. It opens the possibility of security and priority rules in an end-to-end mechanism [41]. Nowadays there are several implementations of Software-Defined Storage and the concept has been a key component in large datacenters.

2.4 Datacenter Operations

Datacenters have evolved a lot since the times of the large computer rooms that housed the mainframe computers. Being the basis of IT in an organization, they play an obvious critical role. In the last decades development has been made from the building construction and design to the practices and policies applied to manage such complex systems. In the following chapters, it will be explored some key concepts in the modern life-cycle of development, management, and operations of IT systems.

2.4.1 Cloud Model

For an organization, a datacenter has a large footprint in terms of consumed resources. In [42] it can be observed that the costs of operation are mainly divided between servers (45%), cooling and power distribution (25%), and power usage (15%). Factors such as management constraints, time, and location, can lead to an unused surplus of capabilities. There is a necessity of a model to promote datacenter efficiency and allow easy resource sharing.

The term cloud computing has been a constant presence in the common vocabulary over the last 20 years. While the everyday notion of *cloud* is quite broad, the American *National Institute of Standards and Technology* [43] defines 5 essential characteristics for cloud computing:

On-demand self-service Automatic provision of available services

Broad network access Capabilities are available over the network

Resource pooling The resources are pooled to serve multiple tenants

Rapid elasticity Capabilities can be provisioned and released arbitrarily

Measured Service Resource usage can be monitored and reported in a transparent way.

All these characteristics facilitate the use and access to systems inside datacenters, therefore helping in more efficient resource use.

2.4.2 Cloud Platforms

There are two different types of clouds, public and private.

Public clouds are offered commercially and are available for a large number of customers. The report [44] reveals two global market leaders: Amazon Web Services and Microsoft, one level below is Google Cloud, Alibaba Cloud and Oracle Cloud Infrastructure, which are also major players. Moreover, many other smaller players offer cloud platforms and services.

Private clouds are owned and operated by private entities, even though they can host several tenants.

Having an on premises infrastructure may have a greater cost but it offers more versatility, control, and security. To deal with sporadic increase in demand for resources there is also the possibility of allocation of more capabilities by resorting to public cloud services. This is effectively a hybrid cloud model where a private cloud is complemented by servers in a public cloud [45].

Large operators often have proprietary management platforms. However, there are some open source alternatives that help the management, control, and provision of the available infrastructure in accordance to the cloud model. Two of the most popular are OpenStack and CloudStack [46].

Openstack was initially developed as a joint project of Rackspace Hosting and NASA. It is currently maintained by the OpenStack Foundation [47] .

Cloudstack was developed by a company called VMOps that was eventually bought by Citrix Systems. The code was released under the Apache Software License and is currently managed by the Apache Foundation [48].

Open Nebula is another interesting platform, it presents itself as a solution to manage enterprise clouds. It is developed by OpenNebula Systems, which offers a proprietary enterprise solution but there is also a free and open source option supported by the OpenNebula Community [49].

Table 2.2 shows that these platforms support a variety of virtualization solutions and techniques.

Table 2.2: Virtualization Support in Cloud computing software

	OpenStack	CloudStack	OpenNebula
KVM	✓	✓	✓
ESXi	✓	✓	Through VCenter
Xen	✓	✓	Community Addon
Hyper-V	✓	✓	✗
Containers	LXC, OpenVZ	LXC	LXD
Lightweight Virtualization	✓	✓	Firecracker MicroVMs

[50–52]

These three platforms support, in one way or another, the same virtualization techniques. They have common goals but sometimes different approaches. The comparative study made in [53] shows that OpenNebula folows a datacenter virtualization approach with the goal of integrating computing, storage, and network solutions, CloudStack aims to facilitate the administrations and accessibility of a large number of virtual machines and infrastructure, and OpenStack helps organizations in implementing a cloud environment with a focus on easy infrastructure provisioning.

2.4.3 Infrastructure as Code

Classical administration tasks, such as setting up virtual computing resources, traditionally involved a lot of manual work. With the increase of the number of machines to manage, automation of a large set of operations is a basic scalability requirement.

The automatic allocation of computing capabilities or services is one important characteristic of cloud computing. Therefore a method for specifying and provisioning resources is needed.

Cloud computing platforms offer Application Programming Interfaces (APIs) to facilitate automatic infrastructure deployment and provisioning. Infrastructure as code (IaC) comes as the process of allocating those capabilities through machine-readable definition files. IaC is not an exclusive to cloud environments.

The adoption of IaC enables the required resources to be specified in definition files. All configurations are done through scripts. In [54] the authors present a few principles that help us comprehend the concept of IaC:

Systems can be easily reproduced It must be possible to easily build and rebuild the whole or part of the infrastructure. Decisions such as hostname nomenclature or software to be installed must already be defined. The reproducible principle means that infrastructure can be spawn effortlessly.

Systems are disposable The whole design of the system reflects the idea that each server is easily created and destroyed as needed. This concept eliminates dependencies on single machines with special configurations. The infrastructure can change, appear and disappear without expected failures.

Systems are consistent Each node in a service must not be unique. Apart from specific differences such as hostname or network addresses, each resource must be an instance of a set of definitions. Multiple identical elements can now be easily created and integrate well with other existing services.

Processes are repeatable All changes and tweaks in the infrastructure must be easy to repeat. This means that even for simple tasks such as a disk resize the process must be implemented in a script or changed in the definition files.

Design is Always changing Dynamic infrastructure must be designed to facilitate change. In overly complex systems it is important to make changes in a safe and quick way. One way of doing this is to minimize the alterations in each step and increase their frequency in a streamlined process.

Infrastructure as code treats infrastructure as if it were software. This opens up for the possibility of implementing the same techniques used in software development.

2.4.4 DevOps

With the generalization of datacenter automation appeared the DevOps concept. DevOps is a set of methodologies and practices that bring software development and IT operations closer together. It is facilitated by the softwarization of infrastructures. The term DevOps comes from the contraction of *Development* and *Operations*.

In [55] it is suggested that DevOps is closely related with the evolution of Agile. Agile is a software development methodology that advocates small release iterations with customer reviews. This eases collaboration, promotes working software, and shortens the development cycle [56].

There are however, different opinions concerning what DevOps is.

Some mention DevOps as a job description with competencies in both software development and IT operations [57]. Others state that “there is no such thing as a DevOps team”, or that it is not a job description nor a department in an organization [58], in any case it is clear that DevOps is on the boundary between applications and infrastructure.

In [55], an interesting framework is proposed by the authors. It divides all concepts concerning DevOps in four categories:

Process The concepts that relate to processes concerned with the outcome and achievements of DevOps related to business processes.

People The culture of collaboration between people, including roles, teams, and communication techniques.

Delivery The core goal of continuous delivery and deployment facilitated by tools that permit versioning, testing, and integration, with automation as a center element.

Runtime Everything concerning the desired expectation of performance, availability, scalability, and reliability.

This approach integrates well the different points of views of each category. This supports Devops as a collaborative and multidisciplinary effort.

One example of this view, is showed in [59]. The authors suggest that at a high level, DevOps can be viewed as a "journey with a set of values that guide behaviour". However for product team members it was more of an end-to-end product view with responsibilities and practices. In the end, more important than those definitions, were well defined goals with clear implementation methods.

2.4.5 Software Frameworks and Tools

Software tools ease the implementation of techniques and procedures in all components of DevOps. Whether they are directly linked to that philosophy or not below are some of the software that are commonly used nowadays.

Git [60] is a distributed version control software that tracks changes in files. It is useful for large teams working on the same code base. It can be used with a remote repository where every developer can push their local changes. Git helps in dealing with conflicts, other features include creation of different branches of the code which helps in feature developing and testing. GitHub [61] and GitLab [62] are platforms that implement remote repositories along with features such as web hosting, project management, code testing and deployment. This makes them overall DevOps enabling platforms.

For the goal of continuous integration and deployment the paper [63] compares too well known and used tools. They are Gitlab CI/CD and Jenkins.

Such tools facilitate deployment between the code base and production. Although they have differences, they share the same generic process.

The code is submitted to a git based repository for production. A trigger occurs in which the tool follows several steps determined in configuration files. The code is built, tested and deployed. If in any step there are reported errors the procedure is interrupted and developers warned. If there are no anomalies the code stays in production.

Configuration and management involves tasks such as installation of software or specific configurations in target computers. A survey and comparison of these type of tools is made in [64]. Ansible uses no agents on the target machines. It makes changes to the nodes over an SSH connection. The tasks are executed according to a file written in YAML, a simple markup language. Chef is another tool mentioned. It uses a client/server configuration. This means an agent must be installed on the target system. Configuration is made in files called cookbooks written in the Ruby programming language. These tools can accomplish in one way or another the same tasks. Ansible having no central node and no agent makes it easier to setup. Chef is older, has more documentation, can handle more complex tasks but has a steep learning curve.

Another category of tools are called Cloud Orchestrators. They use the concept of infrastructure as code to create a desired environment. In [65] there are mentioned several examples of tools like Cloudformation, Openstack Heat, Cloudify, and Terraform. The last two are compatible with a variety of cloud platforms and can request resources from multiple cloud providers concurrently. The authors identify this concept as Sky Computing.

These were just a few examples. Exploring and analyzing the possibilities of each software is a challenge. DevOps created an environment for the emergence of a large number of tools [66]. This is positive and there is still room for more developments.

2.5 Software Architectures

The advancements explored in the previous sections create excellent environments for complex applications to operate. There are some software architectural patterns that can provide some interesting improvements in development and deployment.

2.5.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style where functionalities of a system are separated into services as opposed to being part of a monolithic application. This concept principles are not new, the term's usage dates back to the 90's. Where the concept started being applied by the separation of business logic in a backend server from frontend interfaces. In the beginning of the 2000's the proliferation of web services brought SOA to mainstream adoption [67].

Services usually expose an interface, and can communicate with a defined strict contract or defined protocol. A service might encapsulate a set of functions and can share resources such as a database with other services. To facilitate integration, communication might also be entirely done through an Enterprise Service Bus (ESB) instead of a point-to-point manner.

In [68] it is shown that key concepts enabled by SOA are *reusability* of services and *abstraction* of their implementation. Other aspects are precise specification of functionality, and formal contracts between a providers and consumers.

2.5.2 Microservices

Microservices might be considered a modern take on SOA where services are more fine-grained. Although there is no formal definition for microservice architecture, there are characteristics and patterns that fit this design approach [69].

A microservice is a small program or functional element with a single responsibility. This fact improves maintainability and testing practices, which makes overall development easier. Microservices facilitate scalability and high availability, because more instances of the same functional element can be easily deployed. Communication is made over the network through lightweight technology-agnostic protocols such as HTTP. The objective is to extend on the idea of an environment where every component is loosely coupled. Microservices can be developed more independently and in turn enable a higher frequency of releases and fixes while maintaining high reliability of the complex systems they form. [69, 70]

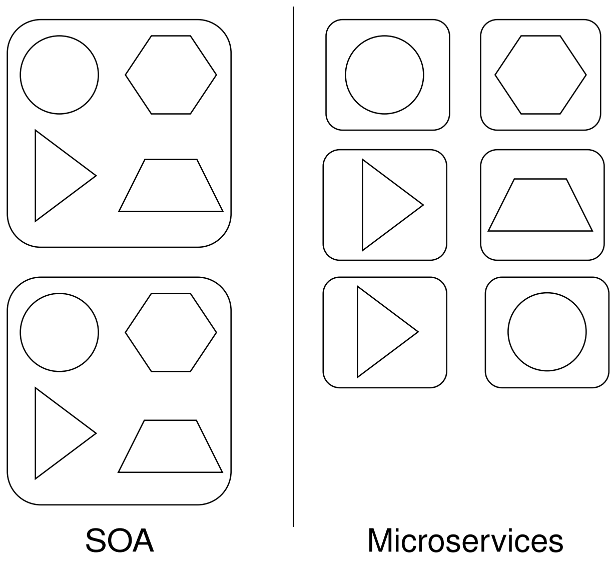


Figure 2.5: Architectural Style Comparison [69]

[71] declares others benefits; an environment of technology heterogeneity, resilience, and ease of replaceability. Figure 2.5 shows an interpretation of the microservices principle given by [69]. On the left services have several contained functions and must be replicated as a whole, on the right each function operates independently as a microservice that can be scaled independently.

In the end these approaches might not be considered strict architectures but a style or paradigm that is used to implement a specific architecture.

2.6 Intelligent Systems

The field of Intelligent Systems is vast. It concerns the study of how systems can be designed to make decisions that reveal learning or problem solving abilities.

In this section we will take a look at the concept of adaptive systems and the application of artificial intelligence and machine learning in IT operations.

2.6.1 Self-adaptive Systems

A self-adaptive system is a system that makes changes to itself and its behavior autonomously at run-time in response to changes in the environment. In [72] the authors identify the following types of Self-adaptive systems:

Type 1 consists in anticipating both changes and the possible reactions at design-time: the system follows a behavioral model that contains decision-points. For each decision point, the solution is immediately obvious given the current perceptions and the acquired knowledge about the environment.

Type 2 consists of systems that own many alternative strategies for reacting to changes. Each strategy can satisfy the goal, but it has a different impact on some non-functional requirement. Selecting the best strategy is a run-time op- eration based on the awareness of the different impact towards these external aspects. Typically the decision is taken by balancing trade-offs between alternatives, based on the acquired knowledge about the environment.

Type 3 consists of systems aware of its objectives and operating with uncertain knowledge about the environment. It does not own pre-defined strategies but it rather assemblies ad-hoc functionalities according to the execution context.

Type 4 is inspired by biological systems that are able of self-modifying their specification when no other possible additions or simple refinements are possible."

Implementation of these types of systems often involves using variations of the MAPE-K model proposed by [73]. A representation of this model is presented in figure 2.6

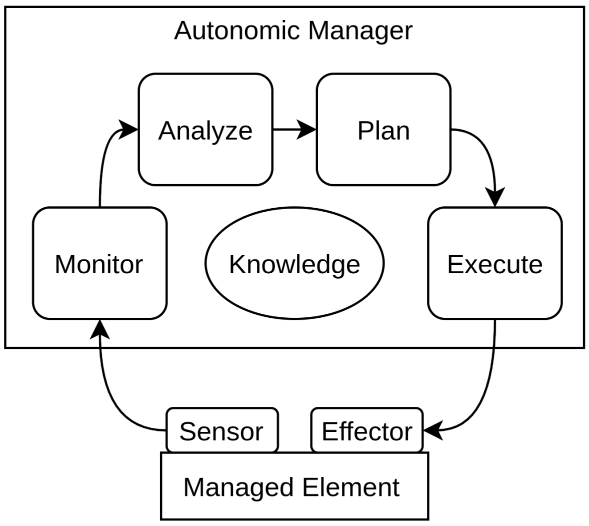


Figure 2.6: MAPE-K Model [73]

For an autonomic manager the capacity to monitor and execute present interesting challenges. Examples are collecting large volumes of data from different sources or integrating the executor with effectors. These aspects can influence the response times of the manager or limit its capabilities.

2.6.2 AIOps

The scale and complexity of IT systems presents a new challenge in its administration. In these systems large quantities of data are generated. Examples are specific metrics, general events, or logs.

There comes a point where it is no longer feasible for humans to supervise and manage an entire ecosystem without help.

Artificial Intelligence for IT Operations or AIOps, is a term coined a few years ago by Gartner [74].

AIOps offers an approach where artificial intelligence methods are used to help build and operate services more efficiently. Implementations of this concept can be applied to anomaly detection, fault source identification or event correlation. Also they can offer optimizations, or improve performance by predicting resource utilization.

In [75] the authors propose a series of levels to assess the degree of automation of AI-supported Administration:

Level 1 The human operator implements the task and turns it over to the computer to execute.

Level 2 The computer helps to determine options. The human operator can choose to follow or not the recommendation.

Level 3 The computer selects the action and implements it if the human operator approves the action.

Level 4 The computer selects and initiates the action and informs the human operator in case the operator wants to rollback the action. If the action fails, the computer informs the human operator.

Level 5 If the computer determines that a previously applied option fails, it repeats the process by selecting and applying a new option from a set of pre-defined, approved activities.

Level 6 If the computer determines that no pre-defined option can be applied, it creates a new remediation or tuning activity.

The authors borrowed and adapted Levels 1 to 4 from a similar model used in autonomous teleoperators. They added Levels 5 and 6, for the goal of achieving a more intelligent and autonomous system. It is expected that in the following years more developments will be made in all levels. The next chapter will present specific cases of systems that fit this model.

Chapter 3

Related Work

There are some recent publications that show applications of the concepts of adaptive or intelligent systems, and datacenter automation. Each one implements, in a way, some goals of AIOps. These examples are shown below along with others works that involve platforms or frameworks that facilitate infrastructure scaling.

Sankie [76] is a project developed by Microsoft, in the Azure cloud environment. It is presented as a service to help developers increase velocity and throughput of changes and bug fixes. Two examples of functionalities are given.

First with a set of changes made by a developer the services can automatically recommend a reviewer for different areas of the source code. Another feature is that it can identify a common subset of changes in files, functions, or modules and associate them to the most number of anomalies.

The data was collected from sources in various stages of the development cycle. Some examples of sources are GitHub, alert logs, services logs, and performance indicators. Sankie uses the environment available in the Azure cloud for storage and computing.

The Azure DevOps service provides APIs that open up collaboration points like GitHub. This way Sankie can ingest events such as pull requests, commits, or test results.

This project continues to be developed. The authors mention that the future direction will focus on increasing the number of analysis and evaluation of the effectiveness of the service.

Analysis of time-series can be an important source of information for capacity planning or anomaly detection. Metrics such as alerts, incidents reports or performance are examples of temporal data that are interesting to explore.

In [77] the authors proposed an anomaly detection service. Their approach consisted in an pipeline with ingestion of data, model creation and alert systems. The techniques that yielded the best results were Spectral Residual outlier detector combined with convolutional neural networks (CNN).

Event logs provide very useful information. However they are unstructured, are written in natural language an often contain unnecessary information.

In [78] the authors created a framework to detect impactful service system problems. Before applying data analysis they used a parser. It removed extra information like file IP addresses or file names. They created log sequences from logs with the same ID and removed duplicate events. After the log data exploration involved clustering techniques and correlation analysis.

Another approach for log parsing was used by [79]. In this case logs were encoded into log keys and a vector of all parameters. They then used deep neural networks to detect anomalies with promising results.

To predict node failure in ultra-large-scale cloud computing platform (Amazon AWS) the authors of [80] used previous alert data of failures in combination with data about the location of nodes. The machine learning technique that got the best performance was random forests combined with oversampling of data from previous failed nodes.

In [81] is implemented an auto scaling platform based on deadlines and budget constraints. Their architecture consists on a decider that collects historical data from a repository, and upon analysis sends a plan to a module they call VM manager. This manager interacts with the Azure Cloud services to scale out servers.

An autoscaling service that enables scaling of applications in a cloud environment is presented in [82]. Applications can be scaled based on defined metrics such as CPU or memory consumption. The architecture is based on microservices. This way scaling monitoring and controller are separated and communicate with API calls. A test implementation is shown working for the cloud platform IBM Bluemix, now called IBM Cloud.

Chapter 4

Solution Architecture

The main goal of this thesis is to contribute to the dynamic management and automatic configuration efforts in a datacenter environment. The proposed solution consists of a platform where data is collected, analyzed, and leveraged to implement adjustments on certain systems. It is an application of the MAPE-K control loop, where the analysis, planning, and execution functions exist inside a platform. The name chosen for this new platform is *OpsManager*. It will help in configuring and manage systems that from now on will be refereed as *target service* or *target system*.

Figure 4.1 shows how the platform fits into a datacenter environment. The *Infrastructure Platforms* element represents the collection of systems that support infrastructure creation, management, as well as development. Some examples are cloud platforms or repositories where *infrastructure as code* is stored and deployed. They support the lifecycle of a target system. *Data platforms* are services that store event and log data generated by the target system. OpsManager integrates with these existing systems by consuming data and generating configuration changes, thus closing the cycle of adaptive infrastructure. The next sections will describe OpsManager in more detail.

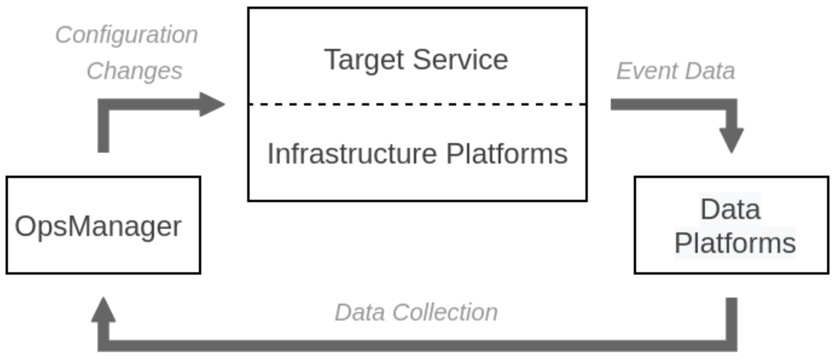


Figure 4.1: Proposed Solution Architecture

4.1 OpsManager Platform

The OpsManager internal architecture is highly modular. This approach meant decoupling functions and services. Components can be developed to fit the implementation environment. They can also be cloned to provide resilience or for scaling purposes. This design choice is based on concepts taken from SOA and microservices.

In figure 4.2 the architecture of the platform can be observed.

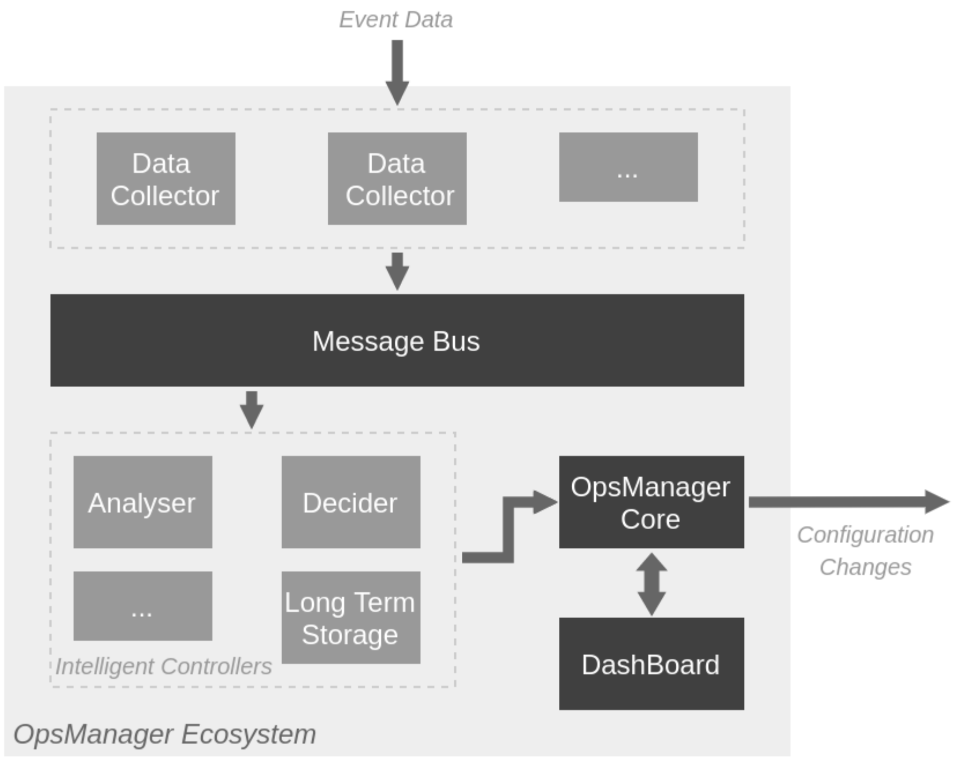


Figure 4.2: OpsManager Architecture

The components include the OpsManager Core for execution of changes, a message bus for module communication, data collectors, the intelligent control group modules for analysis and decision making, and a dashboard that provides a user interface.

4.1.1 Microservices and Message Bus

For a target service metrics or event data generated can come from a few number of sources and they may change over time. For example logs may be sent to a storage location, response times collected by a service, and CPU usage from a third source. This heterogeneous and variable environment is one example that benefits from use of the microservice design approach where a new collector function can be programmed independently for each source.

Data can be analyzed in multiple ways. Concurrent microservices with different implemented logic may consume the same data at the same time. If the data increases in volume and requires parallel processing, there will be a need for partitioning and horizontal scaling. These challenges are easier to approach with a scalable message bus. The possibility of adding new target services, with different data structures and sources reinforces the design choices.

In figure 4.2 each Data Collector on top is a microservice programmed to retrieve data from single source. The other dotted box below with the caption *Intelligent Controllers* encompasses more microservices. An analyzer may implement an algorithm to extract insight from data and a decider might suggest or trigger an infrastructure change. Long term storage may store historical data that will help in forecasting. The dashboard might also receive streaming data and information from the microservices and for example display it in a graph. The microservices should also be able to interact with OpsManager Core via an API.

4.1.2 OpsManager Core and Dashboard

OpsManager core was created out of the necessity of having a way of selecting which configuration changes to apply and providing a backend for a user interface. It has plugins to interact with infrastructure platforms. A decider microservice invokes an endpoint sending infrastructure modifications suggested by its algorithm. OpsManager core then is programmed to wait for user confirmation or apply this automatically. This helps in keeping those microservices fairly simple.

Data visualisation is helpful in conveying the status of a certain system. This is one of the main goals of having a dashboard. The other is enabling a human administrator to have control over certain aspects of infrastructure management. The administrator might be presented with several suggestions of modifications and only apply one, or set a certain decider as automatic and authoritative over a target system.

Chapter 5

Solution Implementation

This chapter details the project design challenges along with implemented solutions. The environment of operation imposed constraints. These are described along with more requirements in section **5.1 Environment**. In section **5.2 Target Service** it is described the characteristics of the system that will be managed. The platform implementation is presented in section **5.3 OpsManager**. Subsections **5.3.1** through **5.3.4** present a detailed description of each component.

5.1 Environment

In the datacenters of Direcção dos Serviços de Informática (DSI) at Instituto Superior Técnico, there are several platforms that create a complex ecosystem. For the purpose of this thesis it was provided a specific environment. The technologies present were required to be used and integrated in this proof-of-concept.

As mentioned before, Openstack [47] is a cloud management platform where virtual machines, virtual networks, storage, among other things can be created and managed. This will be the provider where a target system must be instantiated. Terraform [83] is used to define the infrastructure as code. Besides describing virtual machines, networks, and storage, it can also be utilized for other things such as key generation for each deployment. A centralized secret or token storage platform is available and implemented with the software Vault [84].

Once a VM is created it must be provisioned with Chef. There is a Chef server where configuration guides called recipes can be stored and fetched. All files from Chef and Terraform are version controlled in GitLab. A Group in GitLab may include repositories for the infrastructure code, the provisioning code, the software of a target service, and the deployment definitions with pipelines that create or destroy infrastructure in test and production environments. The is how Gitlab enables *Continuous Integration* and *Continuous Delivery*.

Once a target system is deployed it may be necessary to advertise its existence in a central registry. This service is supported by Consul [85].

The target service might expose a port for incoming external connections. For example a web service listening on an HTTP or HTTPs port. An Nginx [86] server exists in this environment and acts as a reverse proxy. This reverse proxy provides statistics about each backend server such as total requests and response times. There is also an instance of Graylog [87] for the collection of text logs from servers.

5.2 Target Service

The target service to be modified consists of a web application with a variable number of backend servers.

For the purpose of this thesis DSI provided a base template of a target service that included Terraform infrastructure and Chef recipes to instantiate and provision nodes in two datacenters. A GitLab group also existed with deployment pipelines.

A pipeline can be triggered manually, on pushed commits or in other predetermined events. If no changes are made to the pipeline settings or pipeline script each execution will be similar.

Figure 5.1 shows a simple application deployment pipeline. As the name suggests, the “Spin up” job deploys the environment and the “Destroy” job deletes all resources. The objective is to be able to trigger other instances of this pipeline, that is, to “Spin Up” the infrastructure but only make the appropriate differential changes. This means that the new pipeline will initiate without the last having to run the “Destroy” job.

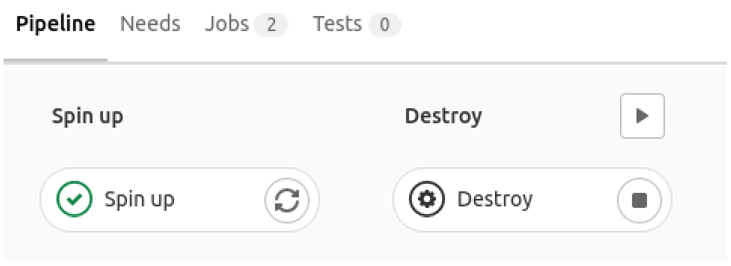


Figure 5.1: Target Service Deployment Pipeline

This is already partially possible with the given target service base template. Terraform is configured to store its state in a backend location accessible to all jobs. Any execution will take the last state and work the changes from there. There were however two details that needed to be addressed. Part of the provisioning of a server included the generation of a policy file for each unique server configuration. The policy file includes all recipes and their versions to be applied. This file is uploaded to the common chef server. If there is a minor change in the recipe or versions, the policy file could be overwritten, or if different names were used there would be a state where new servers that were supposed to be similar to existing ones are actually different.

The pipeline was modified to include verification of existing policies for the same deployment and to not override previous ones. SSH keys for the servers are generated by Terraform inside the job “Spin Up”. If a new pipeline instantiates a new server, it must have the same keys as the old ones. There was the possibility of sharing these artifacts between pipelines. It involved using the “GitLab Job Artifacts API” and having a script retrieve from the last “Spin Up” job. This solution was not optimal, given that it would have to take into account several variables such as branch, deployment version or state of the whole pipeline. Since a central secret storage using Vault was available, it was used to keep keys accessible through infrastructure iterations. Reusing specific variables as a path to the keys there was no need for conditional validations. An example of this path can be: `{service}/{deployment}/{environment}/ssh-keypair`. Every element inside curly braces is a variable: *service* is the name of the application used in target service, *deployment* can be a specific implementation, *environment* distinguishes git branches for testing or production.

Changes to the service can be made in terms of configuration, infrastructure architecture or scaling. This proof-of-concept will concentrate on the latter.

Vertical scaling involves changing the capabilities of current nodes working for a service. Horizontal scaling happens when more nodes are added or subtracted. Environment variables can be set that indicate the number of backend nodes desired. This makes it possible to scale an infrastructure without having to hardcode changes to Terraform files. The template already had this possibility. There was only the need to create triggers to initiate the pipelines.

A web service to be deployed to the nodes was programmed so that each request generates a specific predetermined computational load. Concurrent requests will slow down the average response time.

5.3 OpsManager

This OpsManager implementation materializes the solution proposed in this document. Figure 5.2 shows the architecture implemented along with the environment integration.

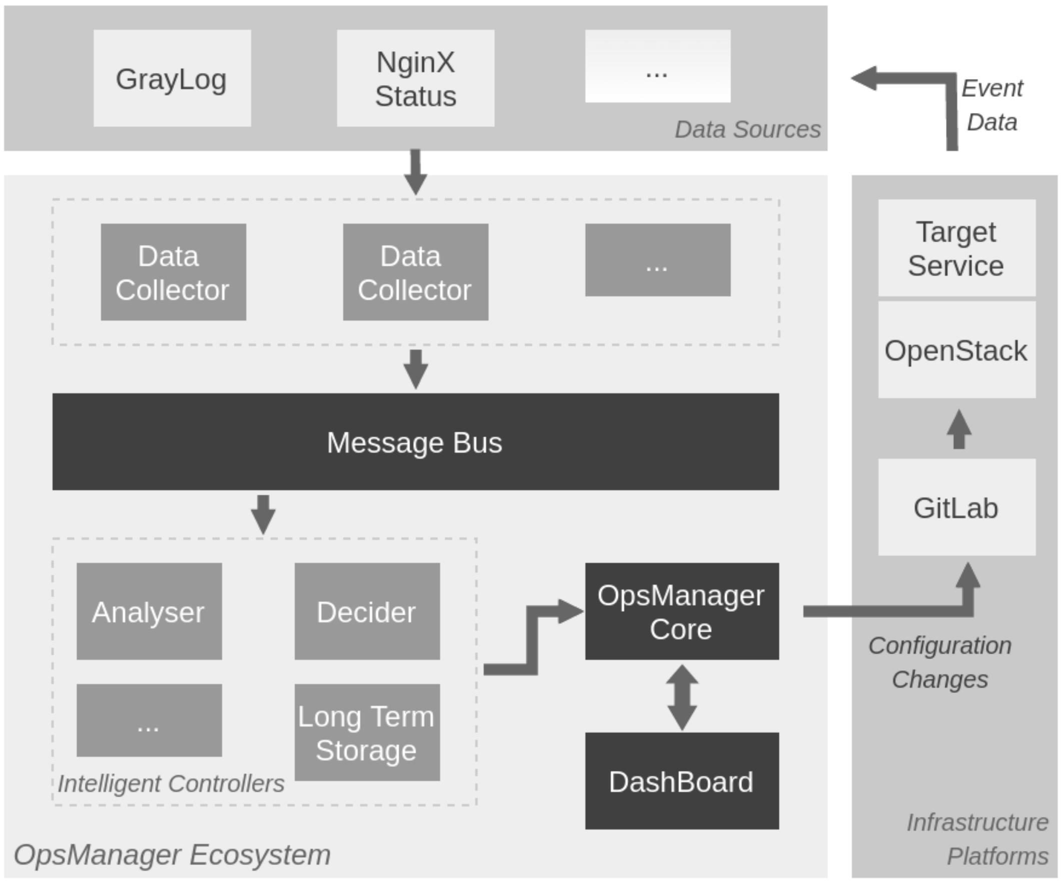


Figure 5.2: OpsManager Implementation

On top several data sources are given as an example of different types of information that can be collected. These can be simple metrics (NingX Status) or text logs and events (Graylog).

On the right side infrastructure platforms that OpsManager may interact with directly. The example given shows the situation previously described in section **5.1 Environment** where GitLab contains code, infrastructure definitions and provision files as well as CI/CD pipelines that create deployments in OpenStack. The arrows demonstrate a flow where information about a service is collected, analyzed and how it generates changes applied to the infrastructure.

5.3.1 Collector

This proof-of-concept will use (NingX Status) as a data source to create the adaptive cycle. It was created a collector that filters and transforms the data as needed.

The NingX Status module contains virtual host status information. The data available is extensive. In an environment with about 50 servers divided into zones, the data can easily reach 200 kB with an average of 4 kB per server.

The collector makes a HTTP request with a predetermined frequency. It receives a json format response. The data is broken down into messages and sent to the message bus. For this specific target service the information needed is the server response times, request counter and error codes. Listing 5.1 displays the information of relevance inside the json response. There is a field called upstreamZones (line 4) that has multiple zones. A zone starts at line 5 and has a list of its servers.

For each server in the field 'requestMsecs' there are two arrays with the timestamps of the responses and the wait time for the response in the array 'msecs'.

In the end current timestamp is added and the final message is sent to the queue in the message bus.

Listing 5.1: Nginx Status Data Strucure

```
1  { 'hostName': "reverse-proxy.example.com",
2    'nginxVersion': "1.18.0",
3    # ... Many more fields
4    upstreamZones: {
5     master.tecnico.opsmgr-targetservice: [
6      {'server': "[fd00::ef01]:80"
7       'requestCounter': 1355
8       'inBytes': 498754
9       'outBytes': 5423423
10      'responses': {'1xx': 0, '2xx': 150, '3xx': 0, '4xx': 2, '5xx': 20}
11      'responseMsecs:{
12       'times': [1636070404950, 1636070604057, 1636070905033, ...]
13       'msecs': [20, 22, 19, 15, ...]
14       }, # ... Many more server fields
15      }, # ... More servers
16      # ... Many more upstreamZones]}}
```

5.3.2 Message Bus

A message bus or queue is a good solution for an environment where data may come from several sources at different rates and read by various consumers.

Apache Kafka [88] is a software that implements such a concept. It provides messaging queuing, storage, redundancy and partitioning of data. It is mostly used for real-time streaming data pipelines for its high throughput and scalability. For this proof-of-concept an implementation of Kafka is used for the message bus module.

5.3.3 Decider

A decider microservice consumes already prepared data and uses it to make decisions about infrastructure. A simple moving average algorithm was chosen. This is an average of the response times of a defined time window, for example the last 10 minutes calculated every minute. It is applied to response times of all servers in an *upstreamZone*. Since the service is a simple web service with a predetermined fixed computational cost, multiple concurrent requests on the same server will increase the average response time.

Algorithm 1 shows a simplified version of the decider logic. Infrastructure changes are separated with a defined interval of **m** minutes. If the moving average of response times, *average_response_time*, is above a certain threshold a scale up operation is suggested. When more node are added to the service the response time is expected to decrease, if not more nodes will be added after the **m** interval.

When a scale up happens, the request counter average is stored in a stack like data structure represented as *last_up_average_requests*. This value is important as the scale down operation cannot be triggered by the decrease in average of response time, after all it is the goal of the scale up. So the algorithm compares the current number of requests with the number of requests that indirectly triggered the last scale up. If the current level is below this number multiplied by an adjusted α parameter it means that the excessive load is no longer present so a new scale down operation is suggested. In the end scale ups have a fixed threshold but scale downs are defined at runtime.

Algorithm 1 Simple Scaling

```
1: while True do                                        ▷ Decider is on a infinite loop
2:     Read messages for s seconds
3:     Update counters and calculate averages
4:     if Last scaling more than m minutes ago then
5:         if $average\_response\_time \geq up\_threshold$ then
6:             $last\_up\_average\_requests.push(current\_average\_requests)$
7:             Add one Server
8:         else if $current\_average\_requests \leq last\_up\_average\_requests.peek() \times \alpha$ then
9:             $last\_up\_average\_requests.pop()$
10:            Remove one Server
11:        end if
12:    end if
13: end while
```

5.3.4 OpsManager Core

The current implementation of OpsManager Core exposes an API. The internal structure of OpsManager Core allows and expects the use of plugins to interact with infrastructure or code platforms such as GitLab. This means that if there is a migration from GitLab to another platform say GitHub, there is only the need to create another plugin. It was created the necessary code to trigger GitLab pipelines upon a simple endpoint invocation.

Chapter 6

Evaluation and Results

The scenario constructed for testing is composed of the target service described in 5.2. This service starts with one backend server. Several HTTP requests are made with the objective of increasing the average response time. The idea was to simulate a usage in working hours with levels of low, medium, and high intensity throughout the day with some level of randomization.

Figure 6.1 shows the response of the system.

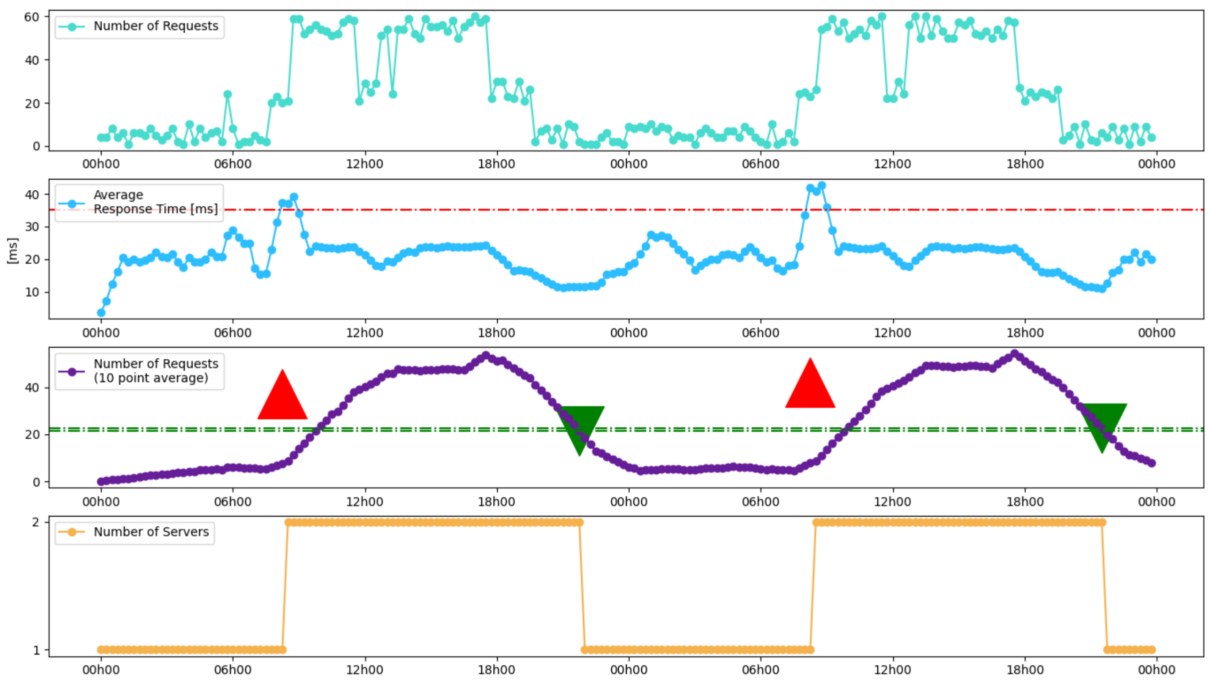
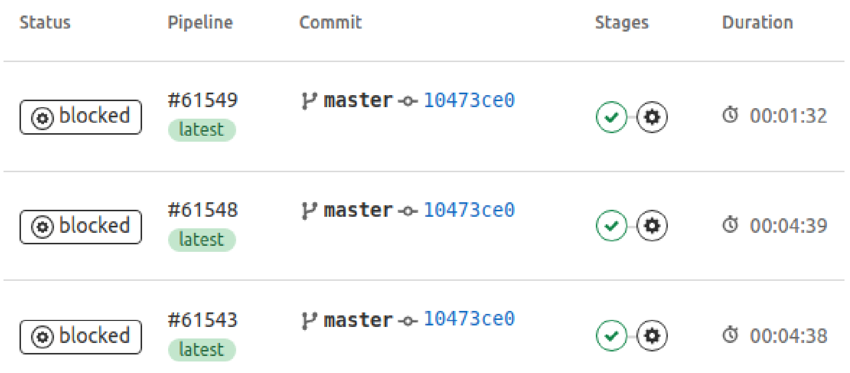


Figure 6.1: Target Service Scaling

The top plot axis shows total requests made in each interval. The second the average response time, which is used to decide scale ups. This threshold is identified by the red horizontal line at 35 ms. The third axis shows the 10 point average of the number of requests. This average decides the scale downs. Since scale downs are computed from the request numbers at scale up they have different thresholds identified by the green horizontal lines. The red upward and green downward triangles show the scale up and scale down moments respectively. The number of active servers at each time can be confirmed in the last axis. The reaction and result of the system can be observed when the load increases. Average response time surpasses the red threshold and another server node is added. In the few next points the average response decreases accordingly.

With these results we can infer that that internal and external integration of OpsManager works as expected.

In Figure 6.2 it can be seen three pipelines ordered with newest on top. All three pipelines make changes to the same infrastructure deployment. The oldest one is the creation of the whole infrastructure. The middle one corresponds to a new server added (scale up). The top one is where the extra server is removed (scale down). The analysis we can take from this result is that with the current model, server provisioning makes the scale up time at 4 minutes which is relatively long. When a virtual machine instance is created a generic disk image is loaded with only the basic updated Operative System. Additional configurations and software installation must be done by the designated chef recipes. This is the main reason for the long times observed.



Status	Pipeline	Commit	Stages	Duration
blocked	#61549 latest	master 10473ce0	✓ ⚙	00:01:32
blocked	#61548 latest	master 10473ce0	✓ ⚙	00:04:39
blocked	#61543 latest	master 10473ce0	✓ ⚙	00:04:38

Figure 6.2: Target Service Pipeline Sequence

Chapter 7

Conclusions

7.1 Achievements

The possibilities offered by the cloud computing model have and will continue to shape the way information systems are implemented and operated.

Developments in artificial intelligence and machine learning will have more and more influence on the way large and complex systems are managed. It is therefore important to have the means to easily implement and test new algorithms and techniques.

One of the goals of this thesis was to provide the first steps towards a management platform architecture to develop an event driven adaptive infrastructure. The developed work accomplished this by designing and proposing a solution that easily interconnects multiple systems and platforms. The results show capabilities for data collection, decision making and infrastructure modification.

7.2 Future Work

The OpsManager platform developed in the scope of this thesis provides the basis of an adaptive, event driven infrastructure.

Although the proof-of-concept presented here provides a model for the overall system, there are several improvements that are advisable to reach an operational state.

The continuation of the development of the dashboard is certainly an important goal. With it comes multi-user capabilities and secure access. Microservices platforms could be used to facilitate their deployment.

For target services a preloaded node image with all the required software might be one approach. Caching of software used in the nodes is another possibility. The use of unikernels or other lightweight virtualization methods instead of whole virtual machines, may provide improvements.

In the end the use of the platform should now make possible to start exploring more complex optimizations algorithms and machine learning models.

Bibliography

[1] P. Denning. Origin of virtual machines and other virtualities. *IEEE Annals of the History of Computing*, 23(03):73, 2001.

[2] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM Journal of Research and Development*, 25(5):483–490, Sept. 1981.

[3] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[4] U. A. Force. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In *Proceedings of the... USENIX Security Symposium. USENIX Association*, volume 129, 2000.

[5] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices*, 41(11):2–13, 2006.

[6] D. Berlind. Etymology of 'hypervisor' surfaces. ZDNet, Aug. 2005. URL `https://www.zdnet.com/article/etymology-of-hypervisor-surfaces/`. Accessed 29 Nov 2020.

[7] A. Desai, R. Oza, P. Sharma, and B. Patel. Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, 2(3):222–225, 2013.

[8] QEMU. Qemu main page. QEMU. URL `https://wiki.qemu.org/Main_Page`. Acessed 2 Oct 2021.

[9] ORACLE. Oracle vm virtualbox. ORACLE. URL `https://www.virtualbox.org/`. Acessed 2 Oct 2021.

[10] VMware, Inc. Vmware workstation pro. VMware, Inc, . URL `https://www.vmware.com/products/workstation-pro.html`. Acessed 2 Oct 2021.

[11] VMware, Inc. Vmware esxi. VMware, Inc, . URL `https://www.vmware.com/products/esxi-and-esx.html`. Acessed 2 Oct 2021.

[12] Microsoft Corporation. Introduction to hyper-v on windows 10. Microsoft Corporation - Documentation, . URL `https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/`. Acessed 2 Oct 2021.

[13] Xen Project . About us - xen project. Xen Project. URL https://xenproject.org/about-us/. Acessed 2 Oct 2021.

[14] I. Habib. Virtualization with kvm. *Linux Journal*, 2008(166):8, 2008.

[15] J. Baldwin. Introduction to bhyve. FreeBSD, Presentations and Papers, 2014. URL https://papers.freebsd.org/2014/baldwin-introduction_to_bhyve/. Accessed 5 Jan 2021.

[16] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[17] S. Garzarella, G. Lettieri, and L. Rizzo. Virtual device passthrough for high speed vm networking. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 99–110. IEEE, 2015.

[18] V. K. Manik and D. Arora. Performance comparison of commercial vmm: Esxi, xen, hyper-v & kvm. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1771–1775. IEEE, 2016.

[19] Microsoft Corporation. Supported windows guest operating systems for hyper-v on windows server. Microsoft Corporation - Documentation, . URL https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/supported-windows-guest-operating-systems-for-hyper-v-on-windows. Accessed 22 Nov 2020.

[20] VMware, Inc. Introduction to bhyve. VMware, Inc. URL https://configmax.vmware.com/guest?vmwareproduct=vSphere&release=vSphere%206.7&categories=2-1,2-2,2-3,2-4,2-5,2-6. 22 Nov 2020.

[21] Microsoft Corporation. Plan for hyper-v scalability in windows server 2016 and windows server 2019. Microsoft Corporation - Documentation, . URL https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/plan/plan-hyper-v-scalability-in-windows-server. Accessed 22 Nov 2020.

[22] Red Hat, Inc. Supported limits for red hat virtualization. Red Hat, Inc. URL https://access.redhat.com/articles/906543. 22 Nov 2020.

[23] Citrix Systems. Configuration limits. Citrix Systems. URL https://docs.citrix.com/en-us/xenserver/7-1/system-requirements/configuration-limits.html. Accessed 15 Dez 2020.

[24] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, pages 275–287, 2007.

[25] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 233–240. IEEE, 2013.

[26] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou. A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 418–429, 2018.

[27] M. Bélair, S. Laniepce, and J.-M. Menaud. Leveraging kernel security mechanisms to improve container security: a survey. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–6, 2019.

[28] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library operating systems for the cloud. *ACM SIGARCH Computer Architecture News*, 41(1):461–472, 2013.

[29] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici. My vm is lighter (and safer) than your container. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 218–233, 2017.

[30] J. Carapinha and J. Jiménez. Network virtualization: a view from the bottom. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 73–80, 2009.

[31] N. M. K. Chowdhury and R. Boutaba. Network virtualization: state of the art and research challenges. *IEEE Communications magazine*, 47(7):20–26, 2009.

[32] P. Goransson, C. Black, and T. Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[33] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[34] P. Goransson, C. Black, and T. Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[35] P. Goransson, C. Black, and T. Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[36] A. Marsico, R. Doriguzzi-Corin, and D. Siracusa. Overcoming the memory limits of network devices in sdn-enabled data centers. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 897–898. IEEE, 2017.

[37] EMC Education Services, editor. *Information Storage and Management*, page 9. John Wiley & Sons, 2010.

[38] J. W. Choi, D. I. Shin, Y. J. Yu, H. Eom, and H. Y. Yeom. Towards high-performance san with fast storage devices. *ACM Transactions on Storage (TOS)*, 10(2):1–18, 2014.

[39] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: integration and load balancing in data centers. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12. IEEE, 2008.

[40] R. Macedo, J. Paulo, J. Pereira, and A. Bessani. A survey and classification of software-defined storage systems. *ACM Computing Surveys (CSUR)*, 53(3):1–38, 2020.

[41] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu. Ioflow: A software-defined storage architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 182–196, 2013.

[42] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks, 2008.

[43] P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011.

[44] R. L. Deepak Mohan, Andrew Smith. Idc marketscape: Worldwide public cloud infrastructure as a service 2020 vendor assessment, September 2020.

[45] M. Raghunandan. Hybrid cloud: An inter cloud communication mechanism. In *Proceedings of the 2018 International Conference on Cloud Computing and Internet of Things*, pages 62–66, 2018.

[46] S. A. Baset. Open source cloud technologies. In *Proceedings of the Third ACM Symposium on Cloud Computing*, pages 1–2, 2012.

[47] OpenStack Foundation. Introduction: A bit of openstack history. Microsoft Corporation - Documentation, . URL `https://docs.openstack.org/project-team-guide/introduction.html`. Accessed 27 Dez 2020.

[48] The Apache Software Foundation. Cloudstack's history. The Apache Software Foundation, . URL `https://cloudstack.apache.org/history.html`. Accessed 27 Dez 2020.

[49] OpenNebula Systems. About opennebula. OpenNebula Systems, . URL `https://opennebula.io/about/`. Accessed 27 Dez 2020.

[50] OpenStack Foundation. Openstack documentation: Hypervisors. OpenStack Foundation, . URL `https://docs.openstack.org/mitaka/config-reference/compute/hypervisors.html`. Accessed 27 Dez 2020.

[51] The Apache Software Foundation. Apache cloudstack installation documentation. The Apache Software Foundation, . URL `http://docs.cloudstack.apache.org/projects/cloudstack-installation/en/4.3/hypervisor_installation.html`. Accessed 27 Dez 2020.

[52] OpenNebula Systems. Opennebula overview: Choose your hypervisor. OpenNebula Systems, . URL https://docs.opennebula.io/5.12/intro_release_notes/concepts_terminology/intro.html?highlight=xen. Accessed 27 Dez 2020.

[53] P. Bedi, B. Deep, P. Kumar, and P. Sarna. Comparative study of opennebula, cloudstack, eucalyptus and openstack. *International Journal of Distributed and Cloud Computing*, 06(01):37–42, June 2018.

[54] K. Morris. *Infrastructure as code: managing servers in the cloud.* " O'Reilly Media, Inc.", 2016.

[55] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.

[56] H. B. Christensen. Teaching devops and cloud computing using a cognitive apprenticeship and story-telling approach. In *Proceedings of the 2016 ACM conference on innovation and technology in computer science education*, pages 174–179, 2016.

[57] J. Smeds, K. Nybom, and I. Porres. Devops: a definition and perceived adoption impediments. In *International conference on agile software development*, pages 166–177. Springer, 2015.

[58] R. W. Macarthy and J. M. Bass. An empirical taxonomy of devops in practice. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 221–228. IEEE, 2020.

[59] M. Senapathi, J. Buchan, and H. Osman. Devops capabilities, practices, and challenges: Insights from a case study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 57–67, 2018.

[60] Git. Git about. Git. URL https://git-scm.com/about. Acessed 3 Oct 2021.

[61] GitHub. Github about. GitHub. URL https://github.com/about. Acessed 3 Oct 2021.

[62] GitLab. Gitlab about. GitGitLab. URL https://about.gitlab.com/company/. Acessed 3 Oct 2021.

[63] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur. Comparison of different ci/cd tools integrated with cloud platform. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 7–12. IEEE, 2019.

[64] J. O. Benson, J. J. Prevost, and P. Rad. Survey of automated software deployment for computational and engineering research. In *2016 Annual IEEE Systems Conference (SysCon)*, pages 1–6. IEEE, 2016.

[65] L. R. de Carvalho and A. P. F. de Araujo. Performance comparison of terraform and cloudify as multicloud orchestrators. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 380–389. IEEE, 2020.

[66] M. Kersten. A cambrian explosion of devops tools. *IEEE Computer Architecture Letters*, 35(02): 14–17, 2018.

[67] N. M. Josuttis. *SOA in practice: the art of distributed system design*, pages 7–13. " O'Reilly Media, Inc.", 2007.

[68] D. Sprott and L. Wilkes. Understanding service-oriented architecture. *The Architecture Journal*, 1 (1):10–17, 2004.

[69] Martin Fowler. Microservices and soa. martinfowler.com. URL `https://martinfowler.com/articles/microservices.html#MicroservicesAndSoa`. Accessed 1 Oct 2021.

[70] J. Thönes. Microservices. *IEEE Software*, 32(1):116–116, 2015. doi: 10.1109/MS.2015.11.

[71] S. Newman. *Building microservices.* " O'Reilly Media, Inc.", 2021.

[72] L. Sabatucci, V. Seidita, and M. Cossentino. The four types of self-adaptive systems: a metamodel. In *International Conference on Intelligent Interactive Multimedia Systems and Services*, pages 440–450. Springer, 2018.

[73] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[74] Y. Dang, Q. Lin, and P. Huang. Aiops: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019.

[75] A. Gulenko, A. Acker, O. Kao, and F. Liu. Ai-governance and levels of automation for aiops-supported system administration. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2020.

[76] R. Kumar, C. Bansal, C. Maddila, N. Sharma, S. Martelock, and R. Bhargava. Building sankie: An ai platform for devops. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 48–53. IEEE, 2019.

[77] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3009–3017, 2019.

[78] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 60–70, 2018.

[79] M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[80] Y. Li, Z. M. Jiang, H. Li, A. E. Hassan, C. He, R. Huang, Z. Zeng, M. Wang, and P. Chen. Predicting node failures in an ultra-large-scale cloud computing platform: an aiops solution. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(2):1–24, 2020.

[81] M. Mao, J. Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 41–48. IEEE, 2010.

[82] S. R. Seelam, P. Dettori, P. Westerink, and B. B. Yang. Polyglot application auto scaling service for platform as a service cloud. In *2015 IEEE International Conference on Cloud Engineering*, pages 84–91. IEEE, 2015.

[83] HashiCorp. Introduction to terraform. HashiCorp, . URL `https://www.terraform.io/intro/index.html`. Accessed 1 Oct 2021.

[84] HashiCorp. Introduction to vault. HashiCorp, . URL `https://www.vaultproject.io/docs/what-is-vault`. Accessed 1 Oct 2021.

[85] HashiCorp. Introduction to consul. HashiCorp, . URL `https://www.consul.io/docs/intro`. Accessed 1 Oct 2021.

[86] NGINX. Nginx reverse proxy. NGINX Docs. URL `https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/`. Accessed 1 Oct 2021.

[87] GRAYLOG. About graylog. GRAYLOG. URL `https://www.graylog.org/about`. Accessed 1 Oct 2021.

[88] Apache Kafka. Kafka about. Apache Kafka. URL `https://kafka.apache.org/intro`. Acessed 3 Oct 2021.