Federated Cyber Range challenges

Master in CyberSecurity: Corporate Strategies; Master Thesis
Université Libre de Bruxelles
Academic year 2019-2020

Last build :June 1, 2020

Version 1.0

Benjamin Nicodème

`https://nicode.me`

`benjamin.nicodeme@icloud.com`

`benjamin.nicodeme@ulb.ac.be`

Supervisor: Prof. Jean-Michel Dricot

Co-supervisor: Prof. Thibault Debatty

Acknowledgements

I would like to thank my supervisor Professor Jean-Michel Dricot, PhD and my co-supervisor Thibault Debatty, PhD for their support and advice during the realization of this Master's Thesis.

I also would like to thank Jérôme Dossogne, PhD for its implication in teaching which guided me during my Master's Degree to improve my maturity in terms of reflection around CyberSecurity.

Lastly, I would like to thank my close friends and family for their support during the whole period of my Master's Degree.

Contents

Acronyms

A | C | D | E | F | G | H | I | K | L | N | O | P | R | S | U | V

A

AAA
Authentication, Authorization and Accounting. 21, 22, 60

AMD-V
AMD-Virtualization. 31

API
Application Programming Interface. iv, 27, 35, 41

APT
Advanced Persistent Thread. 5, 7

AWS
Amazon Web Services. 9, 10

C

CaC
Configuration as Code. 17, 19, 20, 21, 26, 27, 28, 59

CPU
Central Processing Unit. v, 31, 37, 38, 39, 47, 56, 57

CR
Cyber Range. i, iii, vii, 1, 3, 4, 5, 6, 7, 8, 10, 11, 13, 21, 22, 25, 61

CSP
CyberSecurity Practitioner. iii, 4, 5, 6, 22, 23

CTF
Capture The Flag. 4

D

DNS
Domain Name System. 45, 47

DoD
Departement of Defense. 3

E

ECHO
European network of Cybersecurity centres and competence Hub for innovation on Operations. iii, 1, 11

EPT
Extended Page Tables. 31

ESXi
Elastic Sky X integrated. iv, vi, 30, 31, 38, 39, 41, 42, 44, 47, 48, 54, 55, 56, 57, 58, 59, 61, 67

F

FCR
Federated Cyber Range. i, iii, 1, 3, 10, 11, 25, 57, 61

G

GCP
Google Cloud Platform. 9

H

HA
High Availability. 16, 23

I

IaC
Infrastructure as Code. 9, 19, 20, 21, 26, 27, 28, 59

Intel VT-x
Intel Virtualization Technology x. 31

K

KVM
Kernel-based Virtual Machine. iv, v, vi, 30, 31, 38, 39, 53, 54, 56, 57, 58, 61, 69, 70

L

LTS
Long Term Support. 38, 41, 46, 53

N

NIC
Network Interface Card. v, vi, 37, 53, 58, 69

O

OS
Operating System. 6, 8, 41, 44, 58

P

PAYG
Pay As You Go. 9, 24

PoC
Proof of Concept. v, 1, 2, 37, 41, 44, 51, 58, 59, 60, 61

R

R&D
Research and Development. iii, 5, 6, 8, 10, 22, 23

RAM
Random Access Memory. v, 31, 37, 39, 47, 52, 53, 57, 70

RDP
Remote Desktop Protocol. 22

RMA
Royal Military Academy. 1, 37, 61

RVI
Rapid Virtualization Indexing. 31

S

SLA
Service Level Agreement. 39

SSH
Secure SHell. 22, 27, 42, 44, 45

U

ULB
Université Libre de Bruxelles. 1, 37, 61

USB
Universal Serial Bus. 9

V

vESXi
virtual ESXi. v, 31, 41, 44, 52, 53, 58, 59

VLAN
Virtual Local Area Network. 33

VM
Virtual Machine. v, 9, 19, 23, 24, 25, 29, 31, 33, 34, 35, 40, 41, 44, 45, 46, 47, 49, 51, 52, 53, 54, 58, 59, 61

VNC
Virtual Network Computing. 22

VPN
Virtual Private Network. 10, 15, 16, 17, 19, 22, 24, 34, 35, 40, 41, 44, 57, 61

1. Introduction

1.1 Objectives of the thesis

At the time of writing these lines, a few Federated Cyber Range projects and concepts are already existing. Such projects are, however, commercial or at a state or higher level *e.g.* the European level for the *European network of Cybersecurity centres and competence Hub for innovation on Operations (ECHO)* project. Those projects are still in the phase of development or early release, it is consequently difficult to find granular and detailed documentation concerning them.

This Master's Thesis firstly aims to identify the state of the art in terms of Cyber Range (CR) and Federated Cyber Range (FCR).

CRs are a tool capable to simulate large and complex networks and resources. FCRs can be seen as CRs distributed across a federation. More formal definitions will be addressed in the state of the art in chapter 2.

Identify the challenges related the state of the art is also part of the goal of this thesis. The number of challenges is vast and therefore, this thesis mainly focuses on solving some of them such as hypervisor, automation and connectivity.

Finally, the last purpose of this work is to propose a concept addressing the identified challenges. This concept will be demonstrated with a Proof of Concept (PoC). The PoC intends to demonstrate a way to implement shared cyber exercises and training between the Université Libre de Bruxelles (ULB) and the Belgian Royal Military Academy (RMA).

1.2 Contribution

The contribution of the work is firstly to provide a clear list of challenges identified concerning Federated Cyber Ranges.

Secondly, this work also provides a contribution with the concept proposed on chapter 4 and its proof on chapter 5. The limitations and results identified by this work regarding such concept are also part of the contribution.

1.3 Structure of the thesis

The structure of the thesis is the following;

The chapter 2, *State of the Art*, builds the knowledge required to understand the Cyber Ranges and Federated Cyber Ranges and identify the challenges around those concepts. This chapter is essentially the output of the scientific research phase realized during this thesis.

The chapter 3, *Identified challenges*, aims to identify the key challenges around Federated Cyber Ranges based on the State of the Art.

The chapter 4, *Concept proposition*, addresses a concept proposition based on key challenges identified on the previous chapter.

The chapter 5, *Proof of concept*, aims to demonstrate the capabilities of the previously defined concept. The results and limitations of this Proof of Concept are also discussed in this chapter.

The chapter 6, *Future work*, addresses the future work around this concept based on the results and limitations identified during the PoC.

Finally, the chapter 7, *Conclusion*, concludes this Master's Thesis.

2. State of the Art

This chapter aims to establish the *State of the Art* in terms of Federated Cyber Range (FCR). First is defined and explored the definition of a Cyber Range (CR). This chapter then continues with the use cases of a CR. Once the definition and the use cases of a CR are defined, a comparison of the existing technologies and implementations of various CRs is explored. The aspect of federation in terms of CRs is then reviewed.

2.1 Cyber Ranges

The first important step of this work is to define what constitutes a Cyber Range. According to the report of the Australian Departement of Defense (DoD), a parallelism between a shooting range and a CR can be established[1]. The Australian DoD gives the following definition;

Definition 2.1 [1]
"*The word "range" implies an environment for offensive target practice, much like a shooting range for soldiers.*"

A *Cyber* Range would therefore be an environment where CyberSecurity operations could be practiced. The form of these operations could be multiple and will be further developed in subsection 2.1.1.

The definition 2.2 is abstract and does not give any information regarding hardware and physical dependencies of such system. As a matter of fact, CRs can be implemented in a various number of ways. Some of them will be studied in subsection 2.1.4. However, even if the term *Cyber Range* is closer to a concept than a concrete piece of technology, similar components can be identified between different versions and/or implementations. They will be explored in subsection 2.1.3.

Considering the numerous and different existing implementations of CRs, finding the correct way to define them all correctly with a single definition is not an easy operation. Some CRs are specialized in specific tasks and are therefore more dedicated to fulfill a strictly defined function. However, regardless of the specialization they can have, CRs can be seen as an equipment offering similar functionalities as a *sandbox* but at a different scale. A very minimal and generic way to see sandboxes is as an isolated testing environment.[2]

The following definition of a CR will then be used as reference for the rest of this work;

Definition 2.2 [3]
"*A cyber range is a tool that allows to simulate a complete network, and is usually used for cyber training and cybertechnology development. The possibility to simulate large complex networks allows to improve the realism and quality of training and eventually the knowledge, skills and attitudes of cyber specialists. This helps strengthen the stability, security and performance of IT systems used by private companies, governments and military agencies.*"

2.1.1 Use cases

As briefly announced previously, CRs can have multiple purpose. They are generally, non-exclusively, categorized as following [1], [4]–[6];

2.1.1.1 CyberSecurity Practitioner training

It is crucial for CyberSecurity Practitioners (CSPs) (students, professionals, in military or not *i.e.*, every individual implied in CyberSecurity), to keep their knowledge up to date in terms of CyberSecurity. As a matter of fact cyber attacks keep evolving through time. The consequence being that CSPs must constantly keep their level of knowledge up to date to be able to prevent, or at least respond to, those cyber attacks. Therefore, exercises designed to maintaining or improve skills of CSPs can be organized on systems such as CRs.

The training of CSPs is generally realized around scenarios. Those scenarios define the content of the exercises and are required to be designed before the exercises can take place.

Various types of exercises can be designed. The limits of the type and size of exercises depend on multiple variables. It feels common sense that hardware limitation is one of them. The amount of personnel assigned to the management of the CR is also a trivial variable as well at the amount of maintenance required by the CR. In order to have a functional CR a balance between multiple constraints should be respected. Those constraints depend on the needs of the use of the CR. More details on these constraints will be explored in subsection 2.1.3.

Limitations put aside, a common type of exercise supported by CRs is attack/defense as well incident response.[5] Traditional Capture The Flag (CTF) events can also be realized on CRs.

Attack/defense

In attack/defense scenarios multiple teams confront each other. Each team receives a set of compromised or vulnerable machines. The goal of each team is to infiltrate other teams while keeping their machines un-infiltrated and running. To do so, teams must patch their vulnerable machines while finding how to infiltrate machines of other teams at the same time.[7]

Incident Response

Incident response scenarios are in some ways similar to attack/defense except that there is only the defense concern. This kind of scenario is designed to handle incidents in a way to limit damages and reduce time and recovery costs.[8] The difference regarding similarities with attack/defense scenarios is that the attacks endured by teams in incident response scenarios are not dependent on other teams. The attacks are controlled by the system which knows the flaws of the system that defense teams have to handle and can trigger them easily.

CTF events

Cyber Ranges can also host *traditional* Capture The Flag (CTF) events *i.e.*, Jeopardy CTFs. This kind of CTF is traditionally based on challenges that teams have to solve to score.[7] Some of the challenges can require machines to host vulnerable services or challenges to be solved.

2.1.1.2 Penetration testing

Another reason to use CRs is for penetration testing. The United States Department of the Interior gives the following definition regarding penetration testing;

> **Definition 2.3** [9]
> "*Penetration testing is a controlled attack simulation that helps identify susceptibility to applications, networks, and operating system breaches. By locating vulnerabilities before the adversaries do, you can implement defensive strategies to protect your critical systems and information.*"

The implication of CRs in penetration testing or pentesting can be double.

First, as discussed in subsubsection 2.1.1.1, CRs can be used to train CSPs. As CRs can be seen as a giant sandbox[4], using them can help improve pentesting skill of CSPs without impacting any production network, application or system.

The second implication of CRs in pentesting is not about the entity performing the pentest, but about the object of this test. As it is given by the definition 2.3, the objects of penetration testing are applications, networks and operating systems. A broader definition is that pentesting is the practice of testing a computer system.[10] This being said, CRs can help to provide an adequate environment for pentesting a vast amount of computer systems. This environment can be designed in line with the requirements of the penetration testing and the targeted system of this test.

2.1.1.3 Malware analysis

Over the last decade, cyber attacks have evolved to be more complex and smarter than ever before. Some of the most complex and persistent of them are called Advanced Persistent Threads (APTs).[11] APTs are complex malware that can escape most detection mechanisms due to their slow progression rate and low activity. Some of them can stay undetected for years. This type of malware, once detected are difficult to examine due to their evasive techniques. Those techniques are designed to avoid detection of most of traditional detection systems. This can be done in several ways but are out of the scope of this work. However, to cite one in particular, evasive techniques can take the form of activity detection. This means that the payload of the malware will only be triggered if some particular activity is observed on the infected system by the malware.

This kind of evasive techniques, whatever they may be, are difficult to circumvent. This is also due for example to the fact that APTs are targeted attacks. Another example of evasive technique can be that the payload will be triggered only in a particular environment, and more specifically the environment of the targeted entity. This makes the job of defensive and preventive solutions even more complex.

This being said, CRs can help at least in the process of analyzing malware samples, part of an APT or not. Indeed, CRs, due to their ability to reproduce environments, can be helpful to study those kind of *smart* malware.

CRs can naturally be used to study, for example by performing dynamic forensic analysis, of more *standard* malware.

2.1.1.4 Research and Development

Cyber Ranges can be used as well in Research and Development (R&D). R&D can take various forms and some of them overlap with the previously discussed use cases. The idea behind the role of CRs in R&D is, once more, to be able to deliver an isolated environment. This isolated and controlled environment can then be used to test new technologies and new methods of CyberSecurity countermeasures to test their reliability.[5]

In some way, certain aspects of the objectives of this testing process can be similar to some of penetration testing. If a network security solution is studied or pentested, the goal of such approaches is to detect flaws in the system and correct them. One of the differences between penetration testing and R&D, in this case, is the tools available for the testing. Indeed, penetration testing is sometimes called *ethical hacking*, which does not always provide the answer of how to fix the flaws found during this process. If the issue is publicly known, there is much chance that there is already some known procedure to correct it. On the other hand, if the issue is not known it could sometimes be very difficult, in complex environment, to understand and detect the source of the problem. This is where CRs can provide some clarity. One of the particularities of CRs used for R&D is that they generally have embedded logging and correlation systems which can be very helpful in complex testing situations.

2.1.2 Where

Cyber Ranges are used by multiple types of institution *e.g.* governments, military agencies.[3] However, they can also be used by any entity willing to train CSPs.[1], [5]

Commercial CR solutions exist. Some are open source[12] or part of a bigger project. The only limit is money and/or time necessary to build such project.

2.1.3 Requirements

Once again, requirements of such environments highly depend on the intended use of the system. Military CRs will mostly have different set of requirements than the educational ones. And even for the most advanced and developed CRs, a distinction still remains in the objectives of those CRs wherever they are designed to be used. Once more, CRs designed for educational purpose most likely does not have the same set of requirements, or at least not with the same weight over those requirements, than CRs designed for R&D.

Beyond the point mentioned above, which is that the requirements of a solution depend mainly on the use of the system as well as whether the system belongs to a highly secure entity or not, this section aims to establish a list of the functionalities sought in a CR. This list is therefore not exhaustive but contains many of the generally sought-after criteria concerning CRs.

2.1.3.1 Automation

Automation is, or at least should be common to most CR[1], [6], [13]. Multiple reasons justify the need of automation in such systems. First is that static documentation and manual configuration are hard to maintain, error-prone, and rapidly outdated.[6] The other trivial reason is that it is not feasible to manually deploy machines at scale when the number of machines to deploy increases. Therefore, tools for deployment and configuration of machines and services should be as automated as possible. A variety of such tools are already available today. To name some of them; Ansible[14], Chef[15], Terraform[16] tools can help in the automation process.

2.1.3.2 Realism

The realism of the environment can be multiple and can be seen from different perspectives. Virtualization offers various advantages and is highly likely that the environment to simulate has virtualized services or machines. Some other machines, however, like workstations are probably not. Those kinds of machines, servers, workstations, running *standard* OS are the visible part of the iceberg. Indeed, replicating such assets is critical but is, apart from a few details and exceptions, not the most difficult part of the realism targeted.

If we take the hypothesis that the targeted environment to replicate is a nuclear power plant, the management of such critical infrastructure is generally not managed and performed the same way than *traditional* infrastructures.

Indeed, replicate[1] to the perfection of a nuclear power plant and its management infrastructure would probably not be the easiest task to accomplish *e.g.* compared to virtualizing a standard service running on a Microsoft Windows server.

Such realism is important in some cases.[13] One of them, as briefly announced before in subsubsection 2.1.1.3, is malware analysis and more precisely analysis of APTs. The power plant example stated above is not random. APTs had already targeted nuclear environments in the past. Stuxnet, a well-known APT, was designed to target this type of environment. It was targeting specific Siemens industrial systems.[17] This targeting of industrial systems may imply more challenges to be able to replicate such equipment in CRs. Despite the challenge to replicate such systems, this illustrates the need for high realism replication in some use cases.

It is also interesting to mention that the need for realism does not always fit for automation. This being said, a priority in the requirements is probably to establish which are *essential* and which are a *nice to have.*

2.1.3.3 Supported systems

The importance of realistic replicated systems raised the requirement of the amount supported systems in the previous subsubsection 2.1.3.2. A part of the realism is to be able to replicate environments or part of them. However, virtual[2]replication with a reasonable level of realism is not always possible, *e.g.* proprietary, classified technology, etc.

In some cases, this virtual replication is possible. It is then possible to support systems without having to deal with physical hardware and therefore include automation at some point. However, this process, which can be achieved via emulation, sometimes comes with limitations. Even if in the best case, the virtual system does not suffer from limitations and can totally replicate some hardware virtually, there will always have limitations. The most trivial is that it will not be possible to physically affect or test the emulated system. Another one may not come from the result of the virtual system but from the compatibility between the system virtualizing and other systems, *e.g.* the rest of the CR environment or even the CR itself.

Another tradeoff between requirements has then to be found between the level of realism and supported systems.

2.1.4 Types of Cyber Ranges

Generally, when replicating virtually a system, multiple solutions are possible;

Simulation

The Cambridge dictionary gives the following definition regarding simulators;

> **Definition 2.4** [18]
> "*Simulator: a piece of equipment that is designed to represent real conditions, for example in an aircraft or spacecraft.*"

Simulators replicate functionalities even if the underlying hardware and mechanisms are not identical to the original system *e.g.* flight simulator. The result is that only the system may work

[1] Here, the word *"replicate"* does not have to be understood in the process of duplicating the actual environment. This word is used for abstraction reasons and not the mention any of the technologies implicated in this process at this stage.

[2] *"Virtual replication"* abstractly englobes the means of replicating a system without having to physically build the exact same system twice.

in a similar way than the original. Simulation may be sufficient for training but will probably not be for R&D, which most of the time requires realism as discussed in subsubsection 2.1.3.2.

Emulation

The Cambridge dictionary gives the following definition regarding emulators:

> **Definition 2.5** [19]
> "*Emulator: a computer system that is designed to behave in the same way as a different system.*"

Emulators are able to emulate systems and their underlying hardware. This process is helpful when the system to be emulated requires to run on specific hardware. Emulation allows to have a virtual system more similar to the original than simulation. Emulators are well known for their ability to emulate old consoles *i.e.*, to emulate old hardware (that are not produced any more) and the software on top of it. In terms of CRs, emulation may be used to create virtual routers that normally require specific architectures to run. GNS3 is a tool performing this exact function thanks to QEMU, Dynamips and other tools.[20]

Virtualization

When speaking of virtualization, there is generally two types of hypervisor; *type 1* and *type 2*. According to IBM, those hypervisors can be defined as following;

> **Definition 2.6** [21]
> "*Type 1 hypervisors run directly on the system hardware. Type 2 hypervisors run on a host operating system that provides virtualization services, such as I/O device support and memory management.*"

Schematically, hypervisor types are represented as following;

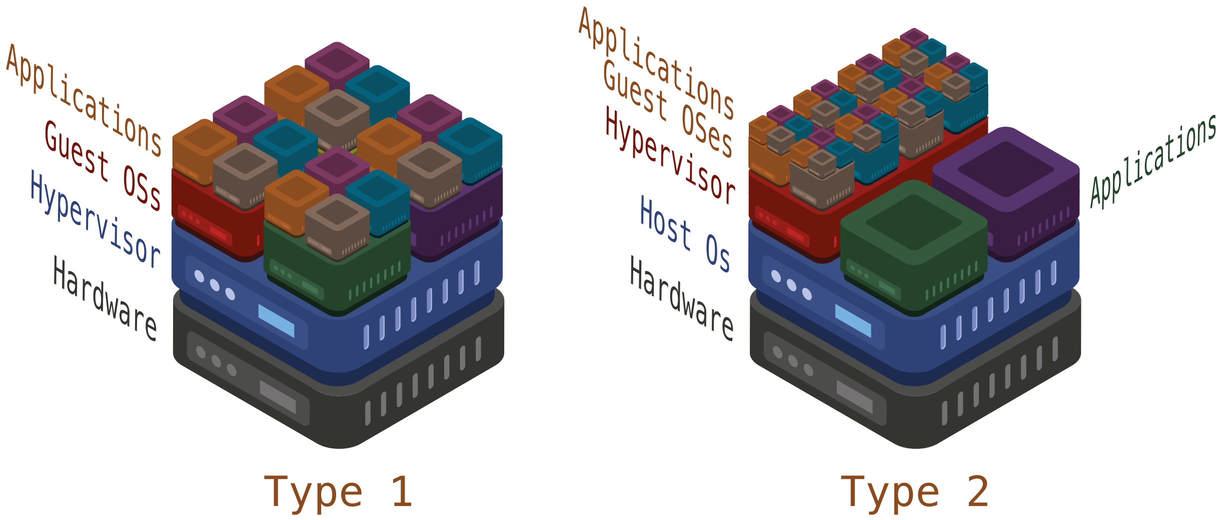


Figure 2.1: Type 1 & 2 hypervisors, based on [21]–[23]

The Figure 2.1 illustrates the differences between the hypervisor types. Type 2 hypervisors are generally commonly found on workstations and type 1 are generally more commonly found on servers. Virtualization is wildly used in development and production environments for various reasons. However, virtualization in terms of CR has limitations. Indeed, the architecture of virtualized systems should be *x86*. Most of OSs used on workstations and servers do not suffer from this limitation. Other architectures are not supported by hypervisors *e.g.* virtualizing Cisco IOS images.

Virtualization and emulation can be combined to circumvent this limitation. Indeed, it is possible to have a Virtual Machine (VM) running an emulator *e.g.* a linux VM running QEMU emulating a raspberry pi which has ARM architecture.

Virtual systems, all distinguished, can be very useful for automation and replication of physical systems without having to own twice the hardware. However, this generally impacts the realism of the virtualized system. In certain cases, this cost is not acceptable and physical hardware is used.

2.1.4.1 Physical

In state-of-the-art realistic exercises, physical facilities and physical hardware are used. *Cybertropolis*, a highly realistic training complex, provides those elements. This complex hosts military exercises with tens of thousands of participants per year.[4] In terms of facilities, the site is 1000 acres, has 300 structures, 70 multistory buildings, 1,6 kilometers (1 mile) of tunnels and over 14,4 kilometers (9 miles) of roads. [4] In terms of hardware, this site disposes of a wireless environment (2G, 3G, LTE), a prison/jail complex, a state-of-the-art smart house, a water treatment plant, and others.

Having such facilities and hardware can be very interesting when speaking of realism. However, when speaking of deployment time, cost, and amount of maintenance, such systems are certainly the worst case possible. Automation can probably be achieved for some part of the systems, primarily virtual ones, but when hardware[3]breaks, it will most certainly take more time to fix, redeploy, or reconfigure than if it was virtual. Since hardware is used, the limit of the supported systems is the hardware itself.

2.1.4.2 Public clouds

Public clouds are interesting for various reasons. Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP) are the leaders in the domain. Even if they have some differences, in this work, we will assume that most of what is possible in one cloud provider is also possible in others and the other way around too.

In terms of public clouds, built-in functionalities like Infrastructure as Code (IaC) can help the automation process. As these kinds of functionalities/services are native to the cloud provider, most of the underlying management and deployments of the infrastructure can be easily performed via such IaC services.

In terms of cost and hardware required, public clouds are interesting *i.e.*, most of the functionalities/services are Pay As You Go (PAYG) and are therefore billed only for what is used. They can also be interesting for entities that do not own hardware. This also means that managing hardware and dealing with hardware issues is not a concern anymore.

However, public clouds are generally as limited, if not more, than traditional virtualization in the number of supported systems. It is possible to have dedicated machines on the cloud. However, limitations will still occur. For example, in a training scenario implicating physical interactions *i.e.*, an infection via USB stick.

This being said, mitigation is possible to counter this limitation. Adopt a hybrid cloud architecture is not always possible and will not always fix the problem, but in some case will mitigate it.

[3]Here, by hardware, is not only intended bare metal servers but hardware like physical routers, switches, etc.

This can be realized via tunneling to the cloud *i.e.*, VPN tunnels.

Existing open source training solutions already exist designed on AWS. EDURange [5][24] and Open-Source AWS Cyber Range [12] are two of them.

2.1.4.3 Private clouds

Private clouds offer benefits of public cloud, including scalability, elasticity, ease of management, with the control and customization of *on-premise* equipment.[25] However, when speaking of private clouds, one of the differences with public could, is that the hardware is not managed by the cloud provider as it is by public cloud providers. The hardware is to be managed by the entity building the private could. This comes with advantages and disadvantages. More customization is therefore possible but at the cost of hardware failure mitigation. Private cloud solutions are for example Open Stack and Open Nebula.[26], [27] Using those allows to create a layer of abstraction in the management of the virtual resources running on the hardware.

KYPO is a CR solution capable of running on both Open Stack and Open Nebula.[5]

2.1.5 Summary

Different technologies and approaches exist to build a Cyber Range. It is possible to combine multiple of them to fill the requirements of the use of the system. This being said, there is probably not a better solution than another *i.e.*, best solution for a purpose can be terrific for another use *e.g.* a state-of-the-art R&D CR requiring huge technical background in CR management may not be the state of the art in terms of training. Technologies and concepts should be picked to fill the requirement gap for the use of the system.

The elements discussed in subsection 2.1.3 and subsection 2.1.4 are not an exclusive list to determine the best solution for the intended use of the system. However, these criteria can and should be considered before selecting the concepts and technologies to be used when building a CR.

As said before, those requirements can take form of the quantity of automation possible, the level of realism compared to production systems or the number of supported systems. A balance should be found between those elements without forgetting the requirements specific to the entity managing the system and to the target audience.

2.2 Federated Cyber Range

The first step is to define the sense of *federation*. The Cambridge dictionary gives the following definition regarding federations;

> **Definition 2.7** [28]
> "*A group of organizations, countries, regions, etc. that have joined together to form a larger organization or government.*"

Federation members, organizations, countries, regions, or any else will be called *entities* throughout the rest of this document. The previous definitions perfectly match the definition of a FCR. As a matter of fact, FCRs are formed by entities who decided to join forces. This could be due to the desire to split costs, to the desire of sharing information, or also the desire to split the manpower required to build and manage an advanced CR.

2.2.1 Cyber Range vs. Federated Cyber Range

The main purpose of FCRs is identical to traditional CRs. However, additional challenges are encountered by FCRs. Those challenges will be discussed in chapter 3.

Traditional CRs can be very complete and sufficient to deal with the reason of their use. However, state of the art CRs are not feasible for each entity desiring one. Indeed, costs related to their creation and management is not always realistic for every entity. In such case a tradeoff between the desired functionalities or capacities of the systems sometimes has to be found.

2.2.2 Components of the federation

A differentiation between building an FCR from scratch or assembling already existent systems to form a federation of CRs is to be done. Indeed, building a new system, a new environment is one thing, but dealing with existent environment is completely different. Both possibilities are interesting and come with advantages and drawbacks. Another distinction can be established around merging existing systems into a federation. Extending homogeneous systems running on multiple different entities is one problem, but merging heterogeneous systems is another one and comes with a lot more of complexity.

2.2.3 Existing federations

Federations used for training and research already exist. Some of them are dedicated to CyberSecurity projects and have similarities with CRs. Such projects are rarely explicitly labeled as FCR. Some of them have similar functionalities and can, in some ways, be considered as such. Others are still in their development phase. It is therefore difficult to find related documentation as they are still in their development process.

2.2.3.1 Emulab

Emulab is an open source emulation software for *testbeds* focused on networking and distributed systems.[1]Emulab can for example use virtualization (via XEN), containerization (via Docker), or use bare metal servers.[29] Bare metal servers, which are *Dell* servers, are reimaged between users.[29], [30] Emulab is using Chef for its deployments and automation mechanisms.[31]

2.2.3.2 DETER

DETER project is based on Emulab. DETER has federation capabilities but the documentation on it does not seem to have been updated for six years, at the time of writing these lines.[32] This can be due to the stop of this branch of the project or just lack of documentation.

2.2.3.3 ECHO

European network of Cybersecurity centres and competence Hub for innovation on Operations (ECHO) is explicitly labeled as a Federated Cyber Range. ECHO, as its name suggests is a European project funded by the *"European Union's Horizon 2020 research and innovation programme"*. According to its website, the project is still under development and has 30 partners from different sectors.[33] The project was officially started on February 25, 2019, and has roadmap dates from 2020 to 2022.[34]

3. Identified challenges

The purpose of this chapter is to identify the challenges represented by the aspect of the federation within Cyber Ranges. Some of them are more detailed than others due to their implication in chapter 4, *Concept proposition*. However, even if all challenges do not have the same level of detail, they are identified which indicates the awareness of their existence. The challenges identified in this chapter are not an exclusive list and it is more than probable that a lot more of them exist.

In this chapter are first addressed the federation and the homogeneity aspects. This chapter then continues with the connectivity of multiple sites. Management and the abstraction are explored thereafter. The chapter continues with the supported systems, the identity management, the access to the machines, the logging and monitoring, the redundancy, the scalability, the resource allocation and finally the costs.

3.1 Federation

As introduced in subsection 2.2.2, a differentiation has to be done between types of federations or at least in their components. The first differentiation to be done is about the initial state of the systems. As a matter of fact, building from scratch or merging existing components share some challenges, but having to deal with existing systems may include more constraints. A second differentiation can be done when merging systems into a heterogenous or homogeneous federation.

Those variations will be discussed in the following subsections;

3.1.1 Building from scratch

Building a federation from scratch implies that multiple entities have a desire to form a federation. It also implies that the build of the federated system will be done without considering the existing respective solutions of the entities (if they have any solution already existing). Building from scratch can be advantageous for multiple reasons. One of them is that the result of the build will probably not suffer from limitations due to the federative aspect as it is the case when several systems are merged. Indeed, architecting the solution before building it, without having to deal with different existing piece of technology can provide a better reflected solution, at least in terms of management. The opposite situation is merging different existing solutions with incompatibilities. If there is no way to run all solutions in parallel with a correct level of integration, a balance must be found. Otherwise, functionalities such as management will be impacted.

Having a federation built from scratch can also help having a better understanding of the components composing the federation. Documentation, as everywhere else, is a painful task that requires a lot of maintenance. If every entity is responsible for its documentation, it means that this same entity will also probably be the only able to document their side of the federation. This also means that if not all entities correctly maintain their documentation less of it will be available when troubleshooting will be required. This could be irrelevant if the federation is never updated and has the same software and hardware running during all its lifetime. In the opposite case, documentation will require to be maintained, which will need updates from all entities.

3.1.2 Merging homogeneous systems

The case of merging same systems to form a bigger one is probably the easiest way of forming a federation, or at least the one requiring the least engineering.

It is not an easy task to draw the line between homogeneous and heterogeneous systems. On some criteria, systems can be similar and on different criteria completely different. Criteria such as management, automation, identity management can help to draw this line.

When systems share the same management, automation and identity management processes, it seems trivial that it will be easier than to design all the system again. However, separate systems may work just fine individually but not anymore when coupled with others, even if they share similar processes.

In the best case, when every criterion match, some work will still be required. Of course, systems will need to be linked together. Connectivity is one of the most important challenges. Indeed, without connectivity, systems will not be able to communicate.

3.1.3 Merging heterogeneous systems

Merging heterogenous systems is probably the most challenging way to form a federation. There are several ways to deal with heterogeneousness. One way is to try as much a possible to reduce the problem to homogeneity. The other way is to deal with heterogeneousness. In such case, if we take the example of incompatibilities at automation level, if no reduction to homogeneity is done, this means more knowledge will be required to maintain and manage the federation. This also means that in some cases some systems may not work correctly and they may require custom fixes.

In other terms, dealing with heterogeneousness may require more time and may induce more troubles with management of the federation. Globally it is probably possible to integrate every system to every other system, but it will also probably require a very large amount of time to do it. Depending on the needs of the federation, it may not be acceptable to have an integration time of 10 years (without even considering the manpower required) to have a functional solution.

Heterogeneous pieces of systems can, however, probably be integrated with limitations. Those limitations can take form of automation, management, etc. This means that it will work, but not as well, or at least not integrated as well as the rest of the system.

3.2 Connectivity

Connectivity is an essential part of the federation. It is trivial that entities should be connected one way or another to be able to form a federation. Connectivity between entities, in a way, can be seen as connecting different sites together. Multiple ways of connecting remote sites exist. Those various concepts/solutions can be compared on various criteria. Some of them are the following;

- Bandwidth,
- Redundancy,
- Expensiveness,
- Ease to deploy,
- Time to deploy,
- Feasibility on long physical distances.

3.2.1 Physical connection

The first way to deal with connectivity is by connecting physically the different sites together. This solution is theoretically feasible, but except for the bandwidth, this solution is the least convenient. As a matter of fact, deploying such hardware will require time, will not be redundant[1], will be the most expensive, the most difficult and slow to deploy, and will not be feasible on long distances. This option is only interesting if the dedicated line already exists between the entities.

To have the best redundancy available, each site will require to be connected to each other. If such thing is implemented, a full-mesh network, will require $\frac{n(n-1)}{2}$ connections[2]. In other words, if 50 sites have to be connected in a full-mesh way, it will require 1225 connections. Without even considering the physical aspect of it (distances between sites, router/switch interfaces, etc.) this will be ridiculously expensive, slow and complex to accomplish. If the physical aspect is added to the equation, doing intercontinental connection is not realistic at all.

In most cases, virtual connections will be the way to go.

3.2.2 Virtual connection

When speaking of virtual connection, Virtual Private Network (VPN) is the well-accepted concept. VPNs are used to virtually link machines. VPNs are most of the time unicast *i.e.*, machine to machine. This link between machines is sometimes called a tunnel. On this tunnel can be added encryption to ensure the confidentiality of data crossing the virtual tunnel. This added layer of security is important because VPN can be used between two machines connected via the internet. Internet being a public network, if traffic crossing the tunnel is not encrypted by the application layer *e.g.* Telnet or HTTP traffic, then traffic will be in clear text. Having clear text traffic going over the internet is not ideal if data is not confidential, but if it is, then it is critical. VPNs can mitigate this issue by providing a layer of security inside the tunnel and then avoid having clear text traffic exchange between entities.

If applications communicating in clear text between entities are required for an exercise or for any reason, then VPNs are more than recommended. The layer of security put aside, multiple ways to connect n entities over VPN exists.

[1]It could be redundant but this will increase complexity, cost, etc.

[2]The formula $\frac{n(n-1)}{2}$ can be seen as each site is connected to all sites but itself, and connections are bidirectional, no requiring wires in both ways.

3.2.2.1 Full-mesh

The most redundant and trivial possibility is to create a full-mesh network between sites of different entities. Indeed, full-mesh architecture provides redundancy when more than two sites are connected. This way if a tunnel is down *e.g.* because of misconfiguration or changes on the tunnel, traffic can still be forwarded to another site and then back to the destination site.

However, with this architecture, the single point of failure will be the VPN endpoints *i.e.*, the machine on which VPNs are setup *e.g.* firewalls, routers, etc. To avoid this single point of failure, HA clustering or equivalent needs to be implemented. The minimal requirement to make this work is to double the number of VPN endpoints. If this HA mechanism is implemented, with the hypothesis that each VPN endpoint has a tunnel to each endpoint except for the VPN endpoint residing on the same site, the amount of required tunnel connections will be much higher than for the previous scenario. If the same example is used with 5 sites with each site having 2 VPN endpoints. If we consider n as the number of sites to connect and m the number of devices per site constituting the HA, then the number of tunnels required will be the following:

$$\frac{n \times m((n-1) \times m)}{2} = \frac{5 \times 2((5-1) \times 2)}{2} = 40 \text{ tunnels}$$

More schematically, 5 sites each having 2 VPN endpoints can be represented as following:

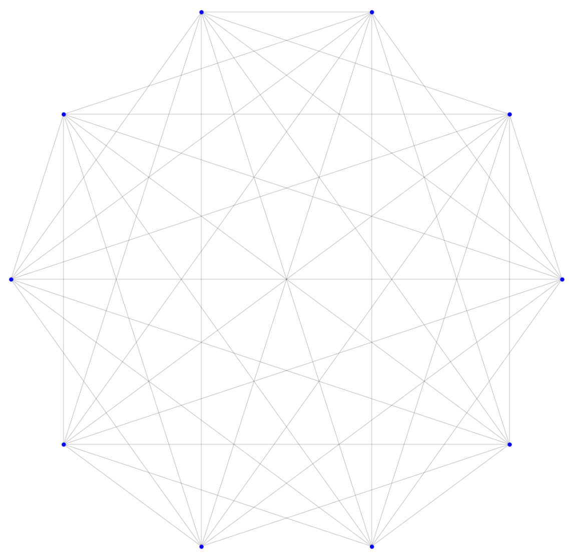


Figure 3.1: VPN tunnels between 5 sites each having 2 VPN endpoints.

On Figure 3.1, the VPN endpoints are represented by the blue dots. The tunnels are represented by the lines between the blue dots.

If the same example is taken but this time not with 5 but with 50 sites to connect, the number of tunnels will be the following:

$$\frac{n \times m((n-1) \times m)}{2} = \frac{50 \times 2((50-1) \times 2)}{2} = 4900 \text{ tunnels}$$

More schematically, 50 sites each having 2 VPN endpoints can be represented as following:

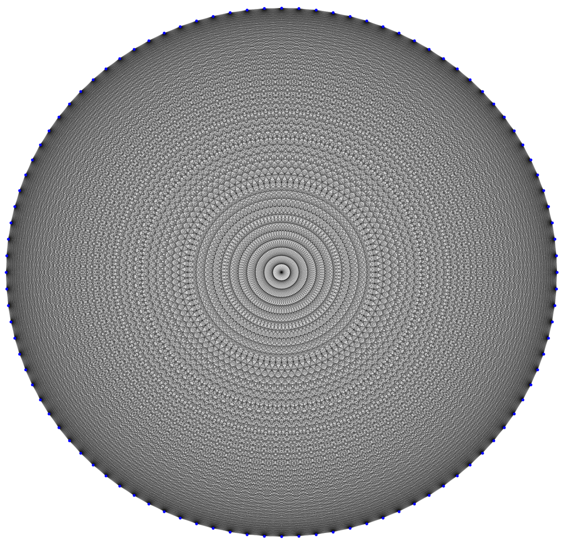


Figure 3.2: VPN tunnels between 50 sites each having 2 VPN endpoints.

Indeed, this difference between 5 sites (10 endpoints) and 50 sites (100 endpoints) gives an indication the function grows rapidly. The function is not linear and can be traced as following:

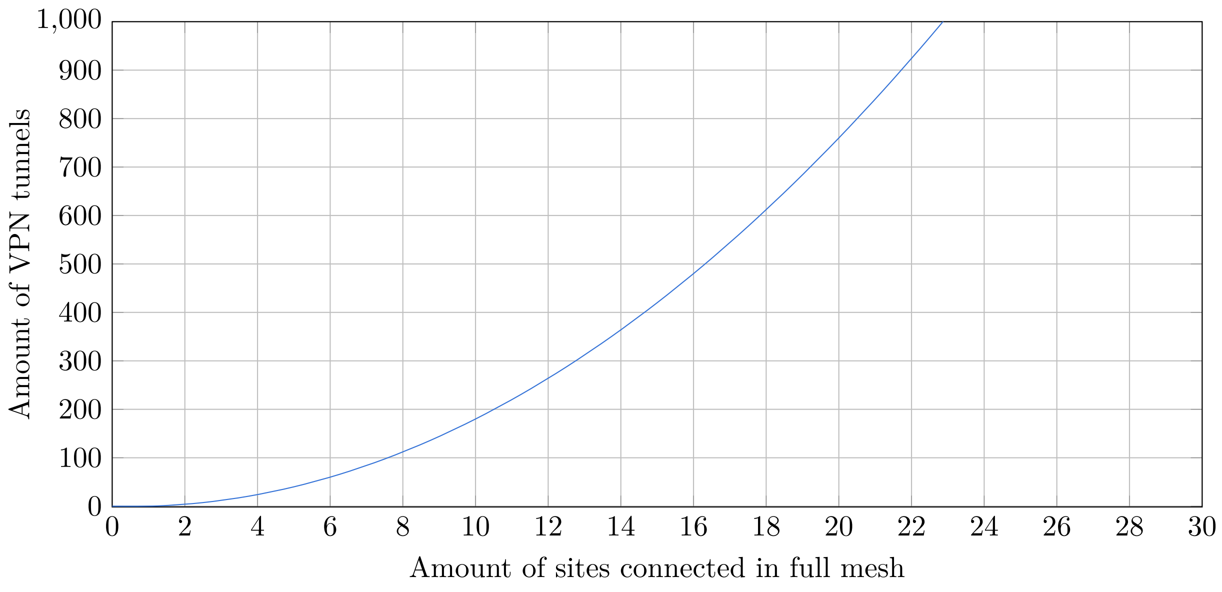


Figure 3.3: $\frac{n \times m((n-1) \times m)}{2}$, with $m = 2$ function.

Depending on the size of the federation, or at least depending on the number of sites to connect, opting for a full-mesh topology may or may not fit. Managing a few dozens of tunnels is a repetitive task but is feasible. In such case, having the possibility to configure the tunnels as Configuration as Code (CaC) is a way to deal with the repetitiveness. It is important to mention that the scalability of this architecture is not as bad as physical connection but is not the best either.

3.2.2.2 Central site

Another solution to deal with connectivity is to use one or multiple sites designated as "central site". This topology can reduce the number of tunnels required compared to full-mesh topology. Indeed, if 50 sites have to be connected, if one site is designated central and 2 endpoints per site are used, it will require the following number of tunnels:

$$\underbrace{49}_{\text{Non-central sites}} \times \overbrace{2}^{\text{Enpoint per site}} \times \underbrace{1}_{\text{Central site(s)}} \times \overbrace{2}^{\text{Enpoint per central site}} = 196 \text{ tunnels}$$

However, having one single central site means that this site will be critical and that the site will have to be up each time the federation is required. Impossible then for this site to shut down. Increasing the number of central sites will make the amount of tunnel tend to the full-mesh function.

Those 196 tunnels can be represented as following:

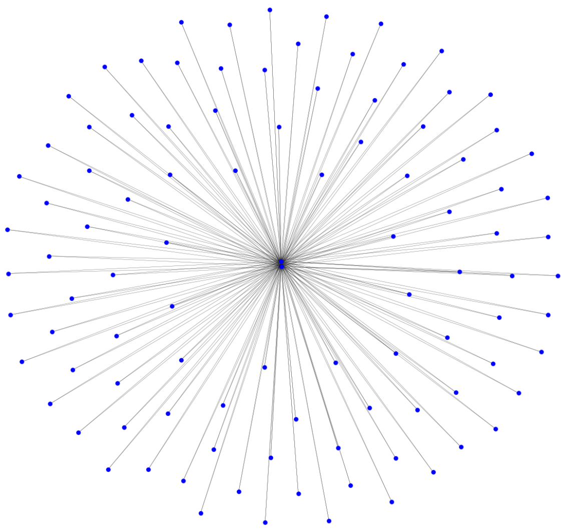


Figure 3.4: 49 non-central sites connect to 1 central site via 196 tunnels.[3]

If the same example with 2 central sites and 48 non-central site is used, the number of sites can be calculated as following:

$$\underbrace{n \times 2}_{\text{Amount of endpoints}} \times \overbrace{c \times 2}^{\text{Amount of central endpoints}} + \underbrace{\frac{c \times 2((c-1) \times 2)}{2}}_{\text{Full mesh between central sites}} = 96 \times 4 \times 2 + 4 = 388 \text{ tunnels}$$

With c the number of central sites and n the amount of non-central sites and the amount of endpoint per site is 2.

[3]To be noted that at the center of the diagram are 2 dots, endpoints

Those 388 tunnels can be represented as following:

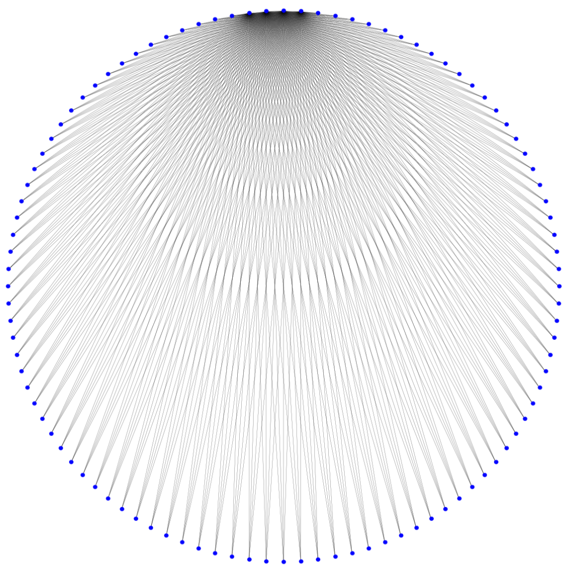


Figure 3.5: 48 non-central sites connected to 2 central sites via 388 tunnels.

This topology requires much less tunnels than full-mesh topology. However, if such topology is implemented, the bandwidth available on the central site(s) also has to be considered. Indeed, as all traffic going from non-central to non-central site will transit through the central site(s), it is recommended that those sites have the best internet bandwidth possible or a bottleneck will be formed on this/these central site(s).

Public clouds

Public clouds generally offer VPN services[35] without having to deal with the bottleneck of an on-premise central site. Redundancy is also assured via 2 VPN endpoints on the public cloud side. This alternative to on-premise central site can also mitigate the issue of the on-premise site always required to be up. The Figure 3.4 represents the topology offered by public clouds.

3.2.3 Type of tunnel

The protocols supported by the VPN endpoints should also be considered.

3.3 Management

Multiple aspects of the management of the federation can be explored. The VMs management, including the way of creating and configuring them with as much automation as possible represents a challenge. Indeed, as explained in section 3.1, when heterogeneousness is part of the federation, it can rapidly become challenging to find a balance between existing systems and still have all functionalities required.

3.3.1 Abstraction level

Another way to deal with automation and therefore Infrastructure as Code (IaC) and Configuration as Code (CaC) in heterogeneous environments will be to have a common abstract language. This language would then be translated in the language spoken by the destination system. This would, however, significantly increase the amount of code required for it to work. Indeed, each entity will have to develop and maintain the connector, or translator, performing the translation from the abstract language to the language used by the IaC and CaC tools of the entity.

Schematically, this abstraction process can be represented as following:

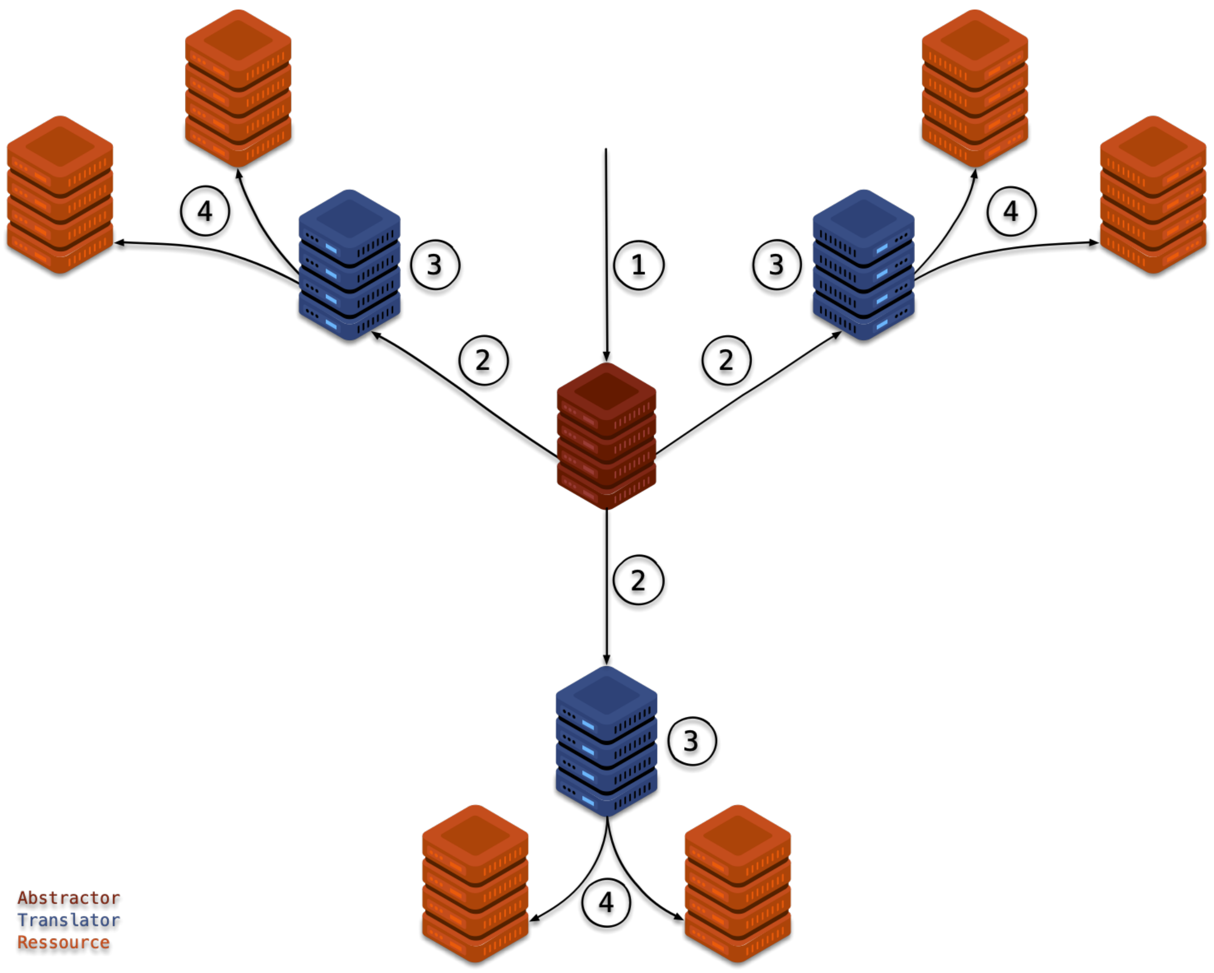


Figure 3.6: Abstraction at IaC and CaC level.

Where the enumeration is the following:

1. Receives an abstract template,
2. Sends respective part of the abstract template to translators,
3. Translates to concrete declaration,
4. Creates resources based on translation.

Once more, this solution will require much more effort to maintain and to develop than having a homogeneous management system dealing with the IaC and CaC. It will, however, provide most flexibility in terms of managing heterogeneous systems part of a heterogeneous federation.

3.3.1.1 Heterogeneous

If the abstraction layer is not possible and merging the existing solutions is the only possibility, multiple tools will need to be integrated to each other. Those will more than possibly suffer from limitations one way or another. The reason is the following;

If the first hypothesis[4]that every piece of software has limitations and/or bugs is taken, then, CaC and IaC being software, have those limitations and/or bugs. If the second hypothesis that not every software suffers from the same limitations and/or bugs is taken, then, CaC and IaC probably have different limitations and/or bugs. The result of combining multiple tools may then result in more functionalities but also more limitations and/or more bugs. Combining multiple tools as a workaround to limitations is one thing, however, combining tools performing the same functionalities to avoid rewriting code cannot produce a positive result.

Indeed combining multiple tools to perform tasks that could have been achieved with a single tool is not the best way to go. For federated scenarios, the best way to proceed will be to have a designated common tool for IaC and for CaC. The designated common tools should only be used with other IaC and CaC tools in last resort.

3.3.1.2 Homogenous

If the management mechanisms/tools are the same, then this case is trivial. In most of the cases, the management can then be extended from one system to other parts of the federation and work properly.

3.4 Supported systems

Supported systems is a challenge more related to CRs then to the federation aspect. As already introduced in subsubsection 2.1.3.3, the number of supported systems is important for realism. In terms of federation, one of the challenges is to improve the number of supported systems that are generally supported by traditional CRs. The federation aspect contains multiple challenges but this one should not be forgotten when building a federation.

3.4.1 Non-Emulable/simulable/virtualizable devices

If for some reason it is not possible to a use a virtual system which is normally physical, then it should be considered to integrate it in its actual working form. Limitations will probably occur, most likely at the automation level. A tradeoff between supporting this system and having full automation should then be considered.

3.5 Identity Management

Identity management is related to authenticate and authorize users and administrators of the federation. Multiple solutions are once again possible to deal with this challenge. One way is to have a central system, redundant if possible, managing Authentication, Authorization and Accounting (AAA).

3.5.1 Central

Despite the chosen protocol used for AAA purposes, if the central solution is selected, entities could choose to fill the AAA system with existing users or external directory services. This solution could help provide a complete visibility of the users. It would also be simpler to have one AAA system than multiple. Indeed, it would be easier to configure only one system on the machines requiring authentication than multiple.

[4]The hypothesis is taken and not explained because demonstrating such hypothesis is out of the scope of this work. With the limitations of the scope of this work, the hypothesis is trivial.

3.5.2 Distributed

A distributed AAA system could avoid having an extra single centralized AAA system to manage. This would, however, restrict each entity to have visibility only on their users. It would therefore probably be more complex to manage users because no one would have a complete visibility on all users. An interesting example of distributed identity management is the Eduroam network dedicated to education and research.[36]

3.6 Access to machines

Ideally, in case of training, access to machines should be possible for advanced users requiring advanced functionalities as well as for beginners. Beginners would probably have more ease to access a web portal opposed to advanced users who would probably not want to be limited to access machines through a browser. For advanced users, access via SSH, VNC or RDP should be possible. Ideally, it should also be possible for users to access the system offsite. Therefore, for advance users, remote access should be implemented and could be with VPN remote access. This feature requires those advanced users to be able to be identified by the remote access VPN. This identification should ideally be based on the AAA system discussed earlier on section 3.5.

3.6.1 Users' credentials

Managing the credentials of the machines and installing appropriate services on those machines are also part of the automation challenge and should be as automated as any other features *i.e.*, as much as possible. This challenge of accessing machines also shares some common points with identity management. Indeed one way to deal with credentials to access the machines could be via the AAA system discussed earlier on section 3.5. This use of the AAA system would require a fine-tuning in the a*uthorization* part of the AAA system. Indeed, authenticating users is one thing, but limiting their access to certain machines is another. Ideally, managing users access should be as automated as possible. However, complexity may appear if the AAA system uses external user directories and cannot deal with their authorization level. In this case, the ideal schema would be to be able to authenticate users maybe on external directories, but also be able to control their authorization level automatically. If not possible, then it would imply that users may have flat level of access resulting in users able to access too much, or not enough resources. This could result in training scenario failure where users were able to access systems they should not and destroyed resources mandatory for the exercise impacting all other CSPs on the training.

If this authorization tuning is not possible, a workaround could be to generate random credentials automatically on creation of resources and only communicate them to users that are supposed to access those resources.

3.7 Logging & monitoring

Logging and monitoring is an important part of a CR. Indeed, loging and monitoring can be very helpful in R&D scenarios to have a global perspective of what is running in the system. Logging and monitoring can also be an important part of training to be able to mark CSPs. Once more the challenge is not directly due to the federation aspect but the federation is increasing the complexity of accomplishing such task. Again, here the possibilities are multiple but mainly, the options are to opt for a centralized or distributed solution.

3.8 Redundancy

Redundancy has been evoked multiple times before with tunnels and connectivity redundancy. Redundancy plays an important role in the federation. As a matter of fact, if critical systems are not redundant and one of them fails, the entire scenario, whether training, R&D or any else, could also tend to fail. In some cases, such failure could impact the reliability of the federation itself. Indeed, if such incident occurs repeatedly, the reliability of the federation could be questioned. However, reliability is important to train CSPs, and is probably even more important for R&D to obtain stable results. Indeed, base R&D results on a non-stable and/or non-reliable system does not make any sense. In worse case scenarios, this could imply that results are not based on what is expected but on unknown or uncontrolled variable and thus introduce a bias on the results making them unusable and/or false.

3.9 Scalability

Scalability in terms of users and integration of new members of the federation is also a challenge. As a matter of fact, if it is planned to increase the size of the federation regarding its users or entities, scalability should not be left out. It is not always trivial to predict how scalable a system would need to be. The best approach is to keep the scalability in mind when picking components to use for the federation. Physical devices can be an issue in terms of scalability. Network bandwidth can also form a bottleneck. Another important aspect of scalability is how complex it will be to add another entity to the federation.

It was discussed in section 3.2 the challenge of connecting entities together. The complexity to manage a large full-mesh topology makes even more sense when a new entity joins the federation. If another entity joins the federation, and if a full-mesh topology is used, this means that each entity will have to create one or multiple new tunnels(based on the HA level) to the new entity. It would require more intervention from administrators of each entity already part of the federation to create those tunnels to connect the new entity. If a central site was used, only its configuration should be updated to add the connection to the new entity. And on the side of the new entity, it would only be required to create tunnels to the central site. This reduces the required number of tunnels to be created to only a few, depending on the HA level. With this central site approach, the number of new tunnels to setup is not dependent on the number of entities already part of the federation. On the other hand, in case of a full-mesh topology, the number of tunnels to be created is proportional to the number of entities part of the federation as seen in section 3.2.

3.10 Resources allocation

If hardware on each entity is dedicated to the federation, resource allocation does not make any sense as the hardware is fully dedicated to the federation. However, even if it is probably not the best option, if the hardware is not dedicated and shares another purpose then it becomes more challenging. Here is why;

First, if the hypothesis that the system used contains an hypervisor, is taken. Then the question of identity management is asked; who can access the system, the VMs? Who can manage the system? The question is more about how to limit access to users from the federation to only access federated resources. Access management will probably increase in complexity.

Second, how to make sure that only VMs are able to communicate between federation resources and not with services external to the federation running on same hypervisor? Indeed, federation resources should no be able to impact services outside of the federation scope.

Last, what about the incompatibilities between federation and non-federation technologies? This, of course, depends on the context of the (non-)federation.

It seems trivial that opting for non-dedicated hardware could drastically increase the complexity of multiple points previously discussed.

3.11 Costs

One of the last identified challenges is about costs. Costs can have different models. Hardware costs are no surprise. They are mostly upfront costs and maintenance costs when hardware breaks. However, if a public cloud is used *e.g.* for VPNs tunnels to connect sites, one of the most popular billing methodology is PAYG. Those costs should be discussed before thinking of using a public cloud in order to avoid falling in an unpleasant situation where costs are too high but the federation requires cloud components.

4. Concept proposition

This chapter aims to describe a concept of Federated Cyber Range including automation, Virtual Machines management, hypervisor management and connectivity between sites. The concept proposed in this work is centered around virtualization. The goal of this concept is to be applicable to existing systems as well as new systems. The heterogeneousness and how the concept proposes to deal with, will be addressed it this chapter.

4.1 Automation

Automation is an essential part of the concept. Ideally, as stated in subsection 3.3.1 and in Figure 3.6, it should be possible to have an abstraction level and a multi-layer automation tool. This would be the best way to proceed. There are multiple ways to add this level of abstraction;

4.1.1 Abstraction + translators

One of them, as stated in subsection 3.3.1, is to have a central server dealing with abstract templates which distributes the relevant configuration to each site to a translator at that site.

The creation and configuration of resources will then be the responsibility of this translator. To fit with the central point of management, those translators should inform the central abstract server of the output of the creation and configuration of resources. This would provide visibility on a single point about the deployment of scenarios dispatched on multiple entities.

For this topology to work, it would be required to at least develop the central abstract server, the translators and their communication protocol.

4.1.2 Abstraction + custom tools

Another way is to have the central abstract server performing the translation and directly creating resources and configuration on different entities. This would reduce the communication overhead between the central server and translator. It would, however, complexify the code of the central server. Indeed, integrating popular Configuration as Code and Infrastructure as Code tools to the central abstract server would be one thing, but integrating custom automation tools would be a much more complex challenge. This part will also be only possible to be developed by the persons having knowledge of the custom automation tools. It would therefore be required that entities work together on the code of this element.

Schematically it could be represented as following:

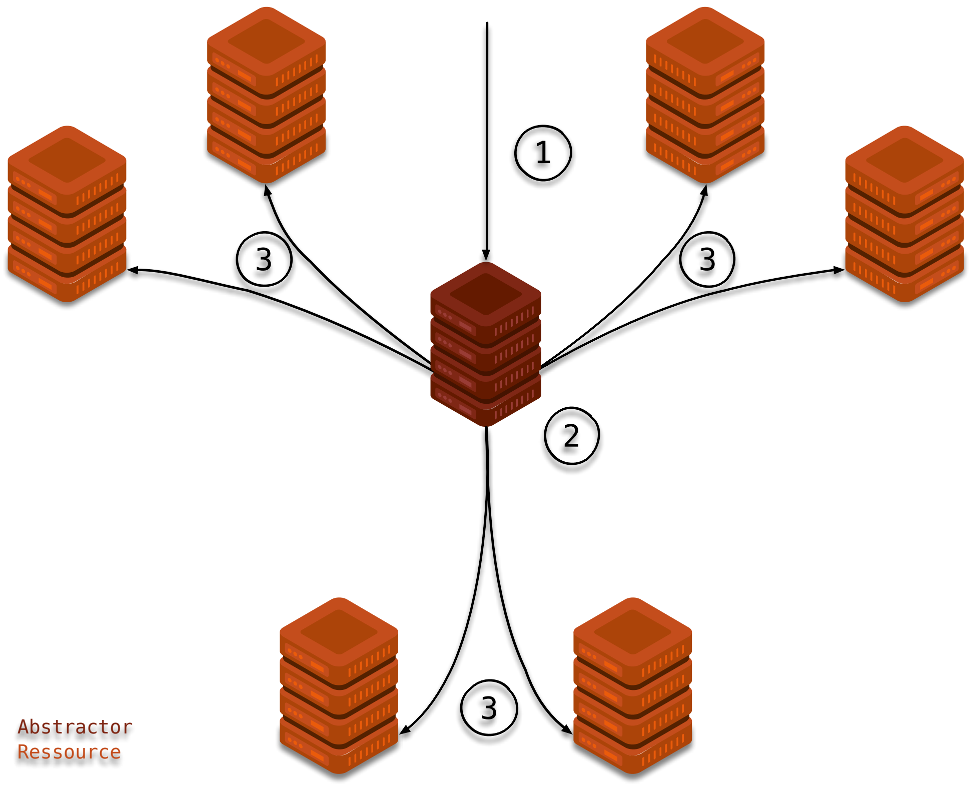


Figure 4.1: Abstraction at IaC and CaC level.

Where the enumeration is the following:

1. Receives an abstract template,
2. Translates to concrete declaration,
3. Creates resources based on translation.

4.1.3 Abstraction + well-known tools

The last method to deal with abstraction is to have abstraction being translated in a well known, documented and publicly available *as-Code* tools. For elements not compatible with the selected tool *i.e.*, custom automation tools, it should ideally be possible to delegate this part of the configuration to a tier server residing inside the concerned entity. Communicating with this server could be done via API calls or SSH.

This solution can be seen as a hybrid solution between having a fully custom automation tool distributed between entities and having only a custom central server dealing with the abstraction. The difference between this solution and the previous one is that the abstraction server will not deal with the custom automation tool. Indeed, the custom automation will reside on a server and will only be triggered by the abstraction server.

Schematically it could be represented as following:

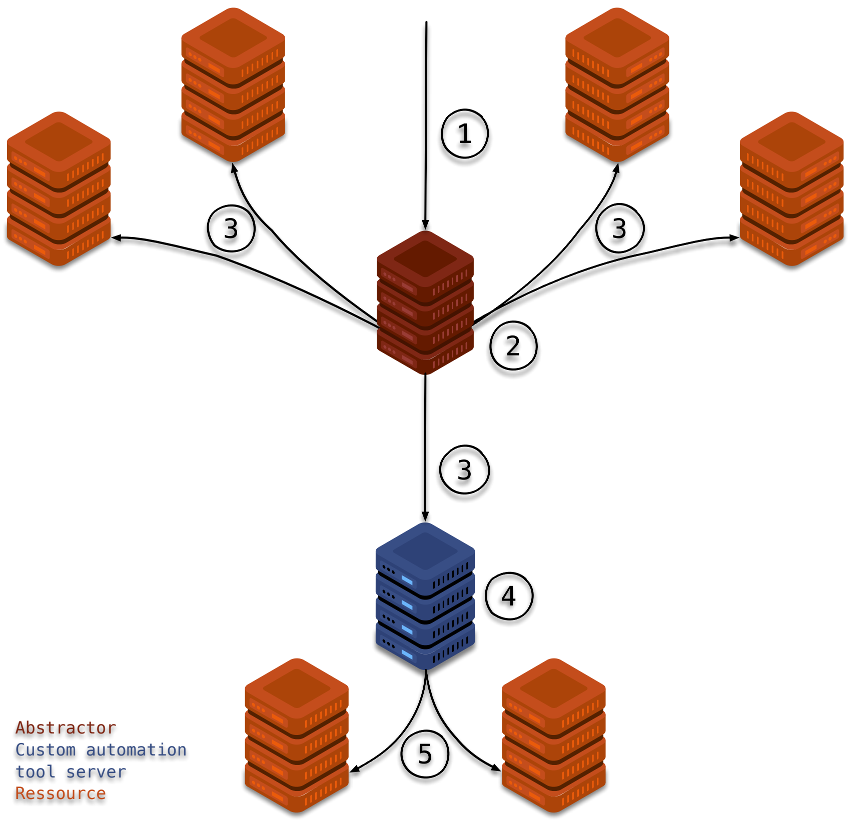


Figure 4.2: Abstraction at IaC and CaC level.

Where the enumeration is the following:
1. Receives an abstract template,
2. Translates to concrete declaration,
3. Creates resources based on translation.
4. Custom resources and automation tools is handled by a delegated server,
5. Delegated server creates custom resources.

This last solution fits better with integration of tools that already exists and will therefore be used in this concept.

To manage Infrastructure as Code and Configuration as Code, multiple well-known tools will be used;

4.1.4 Hashicorp Terraform

Terraform is an agent-less open source IaC tool. Terraform uses a declarative model. This allows the declaration of resources to be created without having to deal with the order of their declaration in the code. Terraform will manage the creation order all by itself. Dependencies between resources can, however, be defined to help Terraform with the creation order. This tool is capable of powerful IaC operations but is sometimes limited in the configuration of the deployed machines. Terraform keeps track of the state of the resources created. Based on the state of resources, Terrafrom is for example able to create a resource graph. It also accelerates the destruction of resources. Terraform can also parallelize the creation and modification of any non-dependent resources. This can be very useful when deploying large scenarios.

Schematically, the way Terrafom work is the following;

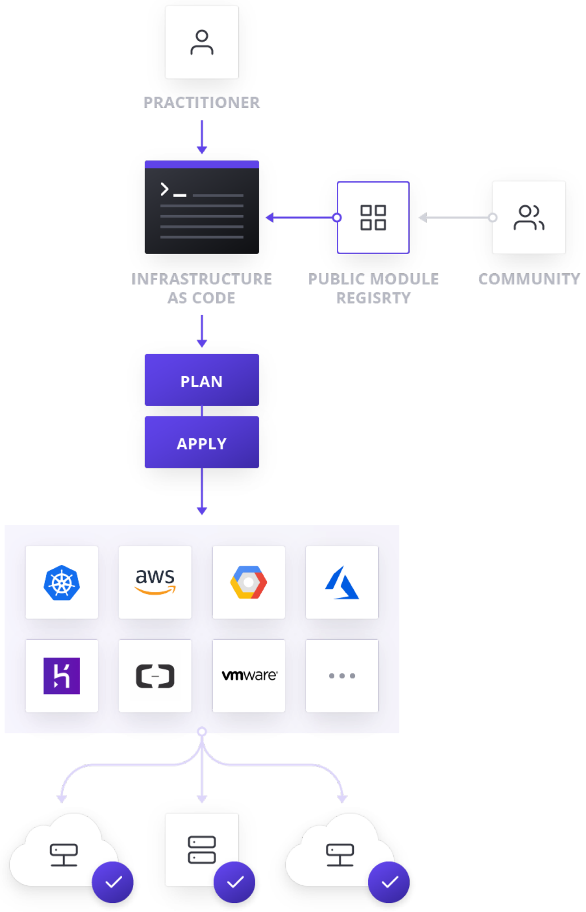


Figure 4.3: How Terraform works.[37]

4.1.5 Red Hat Ansible

Ansible is an agent-less open source CaC tool capable of some IaC. Ansible is a very powerful tool in terms of configuration. Ansible can help circumvent Terraform limitation in terms of configuration.[38]

4.1.6 Hashicorp Packer

Packer is a tool capable to automate the creation of VMs images or templates.[39] It will be useful in this concept to create VMs templates with the minimal requirement of the federation before more configuration is performed at deployment time by Ansible.

4.1.7 Combination of the tools

The previous tools will be used together to fill most of the automation process need. Packer will first be used to create custom images, or VM templates to be deployed by Terrafom and then (re)configured by Ansible when required.

Schematically, the workflow is the following:

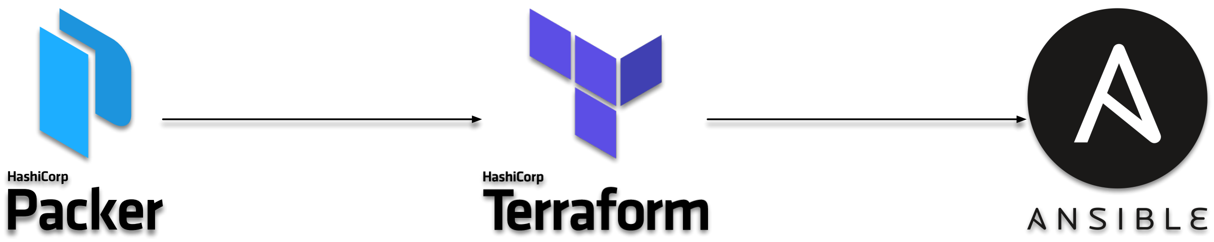


Figure 4.4: Tools combination.[14], [37], [39]

4.2 Hypervisors

As the concept is centered around virtualization, it is trivial that one of the blocks required for this concept to work is hypervisors. Indeed this or these hypervisors should offer various functionalities. Some of them were already discussed in the previous chapters and applies directly to hypervisors. Some functionalities are *nice to have* but will vastly improve multiple aspects such as automation or manageability.

4.2.1 Networking

As already discussed in chapter 3, automation and manageability are required to deal with deployment of exercises requiring large amount of resources. Such large exercises could have as a requirement to connect multiple VMs together. An example is having a virtual firewall and VMs behind it. In such case, like it would be done physically, VMs behind the firewall should not be connected at the same layer than the firewall itself. Indeed, it should be possible to have a virtual switch connecting the firewall to the VM behind it. Virtual switches can be managed in multiple ways but it would be nice to have an hypervisor with built-in virtual switches capabilities. Indeed, having such functionality will decrease the amount of automation required to create topologies using multiple layers of virtual switches.

This feature will also reduce the amount of customization required to isolate VMs. Indeed, virtual switches are the first step to isolate VMs.

4.2.2 Manageability

Manageability is a global and vague term is this context. Indeed, manageability can be found on multiple categories of tasks. Manageability can take form of the way resources are created and manipulated, but it could also concern the way hypervisors are managed. Manageability should always be kept in mind when selecting components forming the federation.

4.2.3 Upgrades

Like every computer system, hypervisors need to be updated for security reasons. This element should also be considered when picking hypervisors.

4.2.4 Functionalities and complexity

Functionalities are required to have a working solution. However, too much of them can increase the complexity of the system. Too much complexity could result in difficulties to troubleshoot problems when they occur. A balance between functionalities and complexity should also be considered before picking any component of the federation.

4.2.5 Hypervisor's selection

Most of today's hypervisors offer more or less the same set of functionalities and fill the previous requirements. However, automation capabilities are not always at the same level. The criteria can be used to make the distinction between hypervisors. Therefore, hypervisors can be compared on the integration with official[1]modules of the tools chosen on section 4.1;

	Official Terraform modules[40]	Official Ansible modules[41]
KVM	0	0 + 3 libvirt modules
Microsoft Hyper-V	0	0
Oracle VirtualBox	0	0
VMware ESXi + vCenter	~50	~150
Xen	0	3 + 3 libvirt modules
OpenNebula	~13	5
OpenStack	~127	~54

Table 4.1: Hypervisors compared in terms of the number of Ansible and Terraform modules.

Having none or a small number of modules is not a good sign in terms of official support. Not having official modules does not mean that the automation tool does not support the hypervisors. It is most of the time possible to find community modules. However, since they are not official, their functionalities may be reduced compared to official ones. It could also be tricky to find correct documentation on those modules.

On the other hand, having a certain amount of official modules does not mean that modules do not have limitations. The number of modules can only be used to determine if there is at least some official modules. If there is no official modules, the integration of the tool with the hypervisors may be limited, inexistent or not possible.

The Table 4.1 contains OpenStack and OpenNebula. However, they are not hypervisors. They can be seen as superior layer adding private cloud functionalities sitting on top of hypervisors.

OpenStack supports the following hypervisors: Ironic, KVM, LXC, QEMU, VMware ESX/ESXi, Xen (using libvirt), XenServer, Hyper-V, PowerVM, UML, Virtuozzo, zVM.[42]

[1]When *official* is mentioned, modules part of official documentation are implied.

OpenNebula supports the following hypervisors: KVM, LXD and VMware vCenter.[43]

This extra layer adds some functionalities in terms of manageability, availability, etc. but with it also comes the complexity to manage this additional private cloud layer. As discussed in subsubsection 2.1.4.3, private clouds also come with complexity.

VMware ESXi + vCenter seems to be most capable solution in terms of "as Code", at least for Terraform (and Ansible) without having to deal with the complexity of OpenStack and Open Nebula. They will be therefore used as the central pieces of this concept.

4.2.6 VMware ESXi

ESXi is a type 1 hypervisor developed by VMware. vCenter is an extra layer sitting on top of ESXi adding functionalities such as central management point of ESXis, VMs migration, VMs templates, VMs clone, etc. This vCenter will provide the manageability required for the management and automation of the hypervisors of the federation.

ESXi is the main hypervisor of the concept. However, to have a sense of heterogeneousness, the concept should also be able to support other hypervisors. Having multiple hypervisors supported by the federation will allow to have a larger number of supported systems. Therefore, the benefit is having more flexibility in the requirements when a new entity joins the federation. In other terms, when an entity that has a hypervisor running that is supported by the federation, it will be easier for that said entity to join the federation with its existing hypervisors than to have to install another one supported by the federation. It is, however, recommended to use ESXi when it is possible. Indeed, using different types of hypervisors will result in an increased complexity of management because other hypervisors will not be able to be managed by the vCenter. As a matter of fact, the vCenter can only manage ESXi hypervisors. If ESXi cannot be directly used as the main hypervisors, the workaround is to use *nested virtualization*.

Nested virtualization, as its name suggests, makes possible to use virtualization embedded in virtualization. In other words, nested virtualization adds the ability to virtualize hypervisors and their underlying VMs. It is then possible to benefit from ESXi and vCenter functionalities while still being able to use the host hypervisors. More specifically, if an entity has only one machine running KVM hypervisor (or any other hypervisors supporting nested virtualization (of ESXi)) and cannot migrate from KVM to ESXi for a specific reason, virtualizing ESXi in a nested VM will provide the solution to still be able to connect this system to the federation.

Nested virtualization requires hardware-assisted virtualization compatibility. For Intel processors, this piece of hardware is called *Intel VT-x/EPT* and *AMD-V/RVI* for AMD processors.[44]

Nested virtualization also adds ability to control resources allocated to the federation. Because the nested ESXi, or virtual ESXi (vESXi), is a VM, resources allocated to the federation can be controlled as a standard VM *e.g.* the amount of CPU, RAM and disk. In the previous example, KVM was the host hypervisor that could not be migrated, with this same example, the resources allocated to the federation can be controlled on the KVM hypervisor. This process can also be used to provide isolation between VMs running on the host hypervisor *i.e.*, the KVM hypervisor, and the guest hypervisor *i.e.*, the vESXi hypervisor.

Due to the additional layer of virtualization, nested virtualization can suffer from performance loss.
The following procedure should be followed when adding hypervisors to the federation:

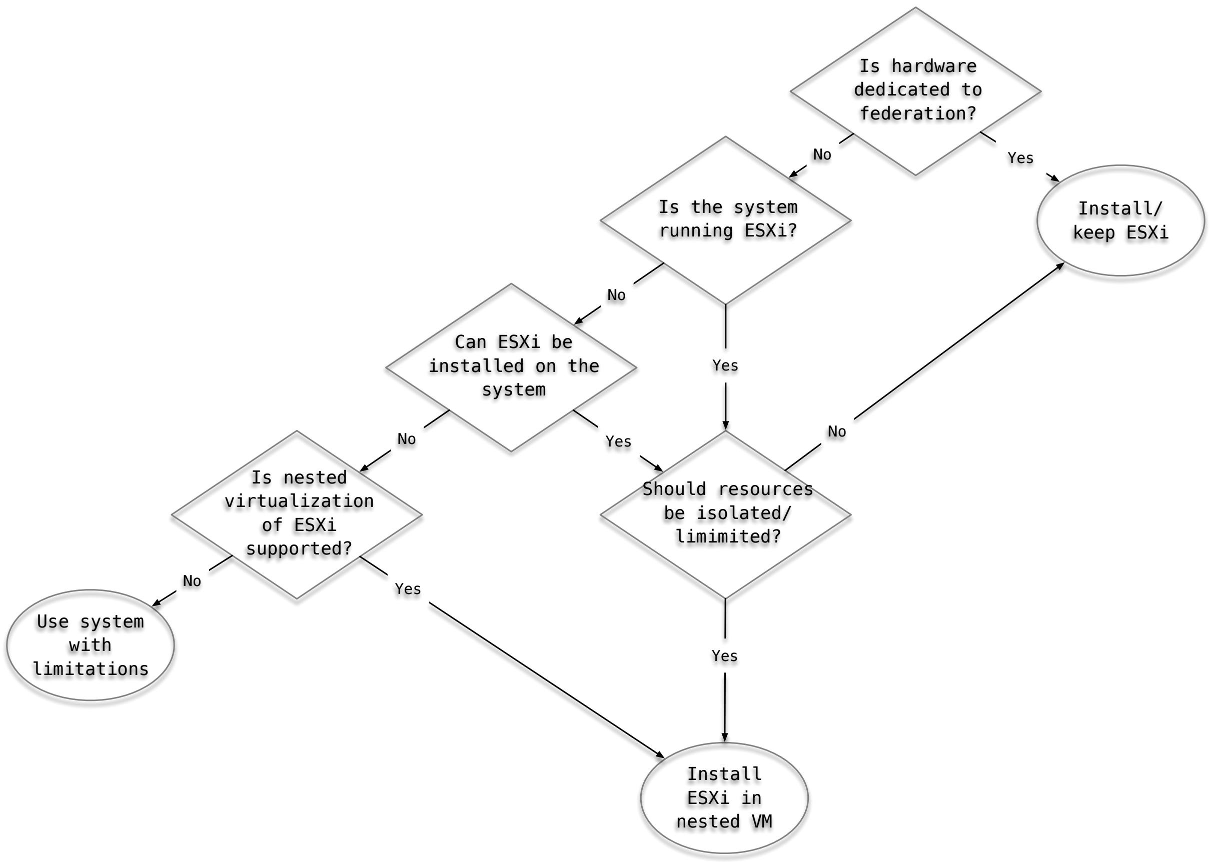

Is hardware dedicated to federation?
No
Yes
Install/ keep ESXi
Is the system running ESXi?
No
Yes
Can ESXi be installed on the system
No
Yes
Should resources be isolated/ limimited?
No
Yes
Is nested virtualization of ESXi supported?
No
Yes
Use system with limitations
Install ESXi in nested VM

Figure 4.5: Hypervisor integration procedure.

At this stage, the concept includes hypervisors and how to deal with the automation required for deployment of resources. Schematically, except for hypervisor and vCenter installations and configurations, it is possible to automatically create the following resources;

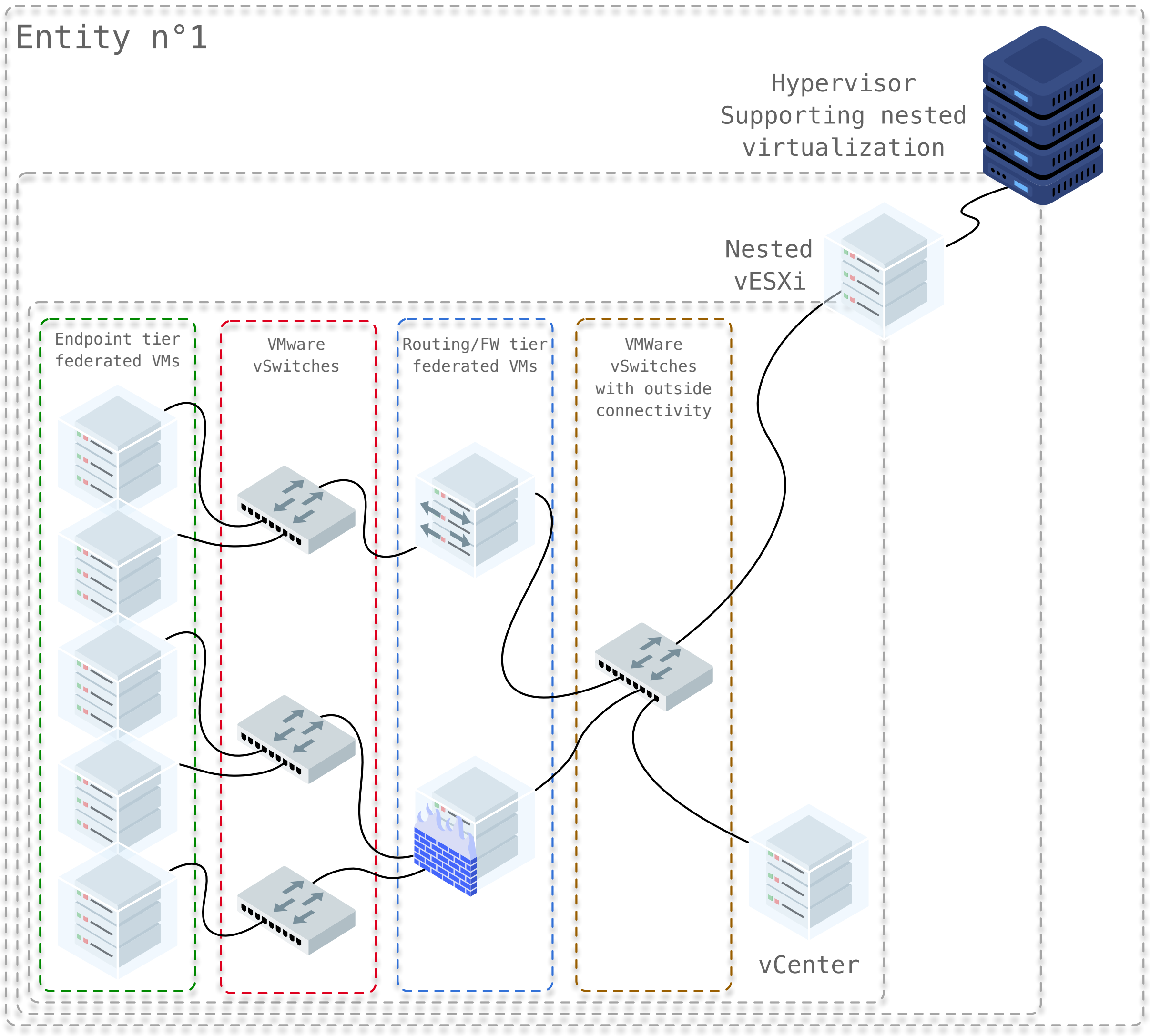

Entity n°1
Hypervisor Supporting nested virtualization
Nested vESXi
Endpoint tier federated VMs
VMware vSwitches
Routing/FW tier federated VMs
VMWare vSwitches with outside connectivity
vCenter

Figure 4.6: Hypervisor capabilities with nested virtualization.

The Figure 4.6 has several VMs and virtual switches. The *virtual switch* is connected to a physical (network) adapter and to the *routing/FW(firewall) tier*. This virtual switch is basically the bridge between the VMs on the routing/FW tier and the outside connectivity *e.g.* the internet. The *routing/FW tier* is where VMs performing firewalling or sub-routing should be placed. Such process can be interesting for exercises requiring such machines *e.g.* trainings or experimentations. Deploying those type of machines can also help segment network access of the *endpoint tier*. The *virtual switches* are connecting the resources from the *routing/FW tier* and *endpoint tier* together. VLANs can also help improve the network segmentation across all tiers.

It is worth noting that when nested virtualization is not required, the guest hypervisor and the host hypervisor are the same machine, removing the nested layer and leaving only one layer of virtualization.
The next element to add to the concept is the ability to connect multiple entities together.

4.3 Connectivity

The next element is about the connectivity between the hypervisors. Two scenarios have to be considered. The first one is trivial and is about a federation formed only by two entities each having only one machine. In such scenario, the best way to deal with the connectivity is with a VPN tunnel from one entity to the other. It is common to find multiple protocol supporting site to site VPN capabilities on current technologies. Depending on the use case, a protocol may fit better than another.

The other scenario is when there is more than two sites to connect. In that case, one central site should be designated. When the load cannot be handled by one of the sites of the federation, moving the central site to a public cloud could resolve the problem. It is suggested to review the estimated cost of such migration before doing it. The protocols supported by the cloud provider should also be considered. Most of the time, IPSec is the only protocol supported by cloud providers for site-to-site VPNs.[45]–[47]

Those two solutions are the simplest to configure and to manage. They should also not have a negative impact on the quality of the connection between sites.

On the hypervisor side, any VM with VPN capabilities and with compatibilities with the selected protocol can be used to assure the VPN connection between sites. This system should ideally also support dynamic routing protocols to avoid static routing between site (at least when more than two sites). Using a VM for the VPN connectivity will put the system under control of the hypervisor part of the federation. No additional hardware will therefore be required. Adding this connectivity VM will impact the topology initially defined by Figure 4.6 as following;

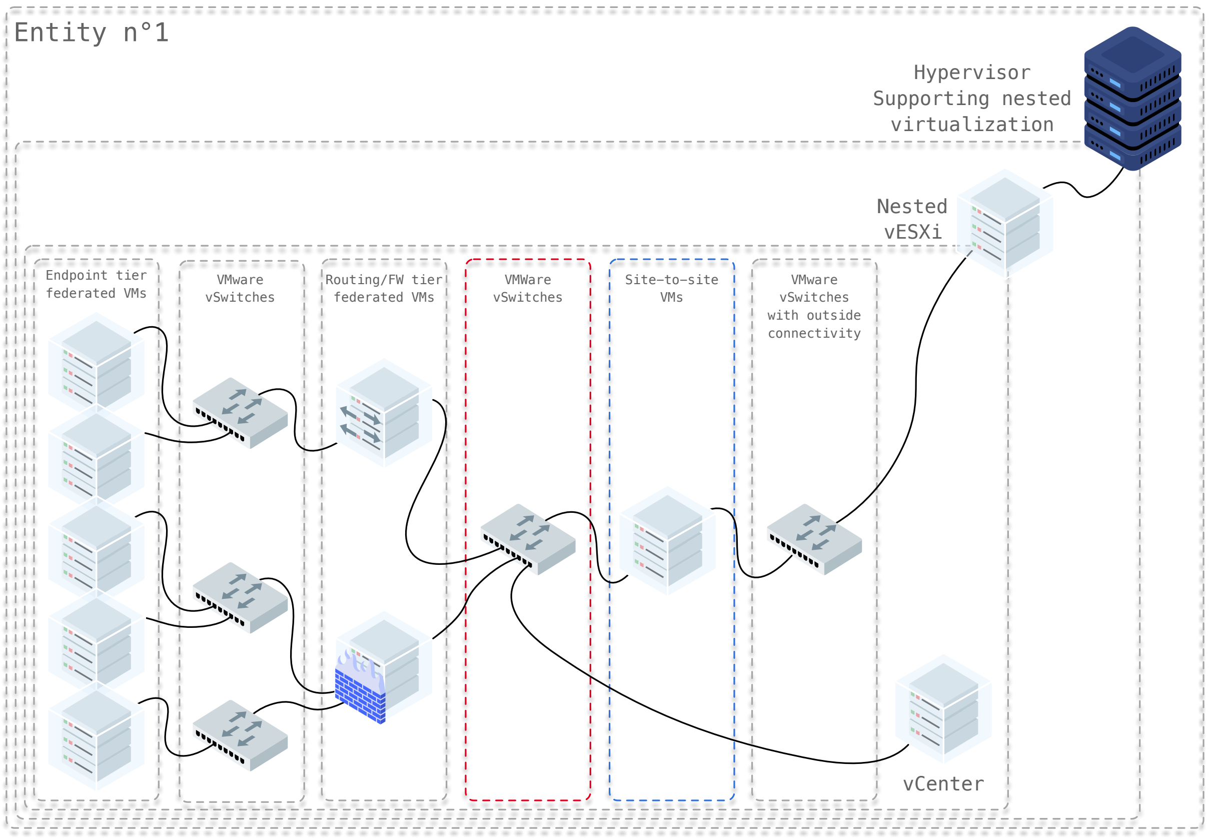


Figure 4.7: Hypervisor capabilities with nested virtualization and VPN VM.

The only different elements between Figure 4.6, and Figure 4.7 is the *virtual switch* and the *site-to-site* VM. The *site-to-site* VM is assuring the VPN connection to the federation.

Conceptually, the sites will then be connected via the VPN VMs as following;

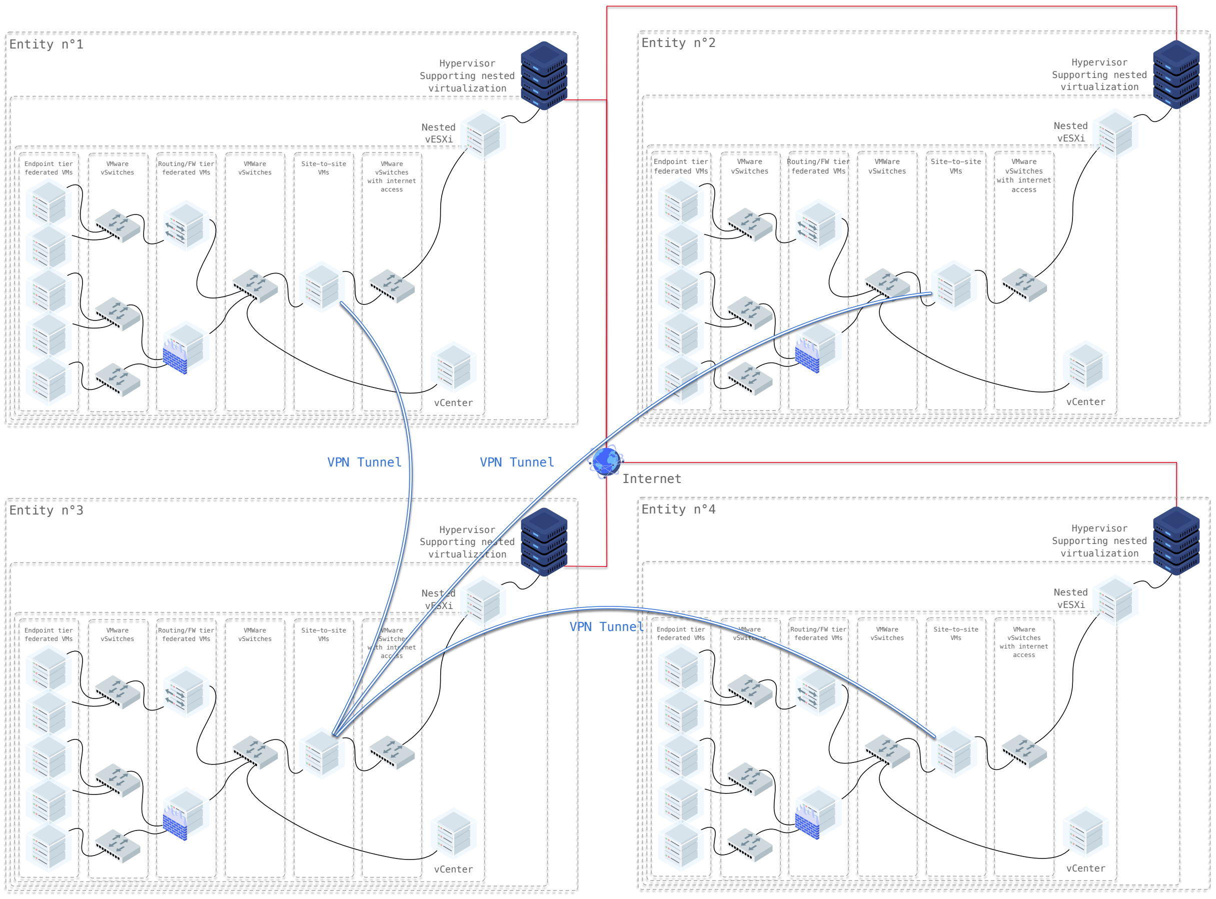

Entity n°1
Entity n°2
Entity n°3
Entity n°4
Hypervisor Supporting nested virtualization
Nested vESXi
Endpoint tier federated VMs
VMware vSwitches
Routing/FW tier federated VMs
Site-to-site VMs
VMware vSwitches with internet access
vCenter
VPN Tunnel
VPN Tunnel
VPN Tunnel
Internet

Figure 4.8: Hypervisor capabilities with nested virtualization and VPN connectivity.

Where the entities are the same on the Figure 4.7 then on the Figure 4.8. The added elements on Figure 4.8 are the *site-to-site connections* and *internet connections*.

4.3.1 Encapsulation

If a specific scenario requires a particular encapsulation, it should be possible to re-encapsulate interesting traffic inside another tunnel with the required encapsulation represented in blue.

4.3.2 API manageable

Like any other component of the federation, it should be better to have a system capable of being configured with API calls or any other automation capabilities. This can speed up the process of adding new entities to the federation. It can also decrease the number of configuration error by using a template created for this purpose. If the selected solution is not able to work with API, VM templates and Packer can help produce a base system and decrease the amount of manual configuration required when adding a new entity to the federation.

5. Proof of concept

This chapter aims to demonstrate the concept previously evoked on chapter 4.

Ideally, this concept should have been implemented and tested in real conditions between the Université Libre de Bruxelles (ULB) and Belgian Royal Military Academy (RMA). However, it was not possible due to COVID-19 situation in Belgium at the time of realizing this work. The following Proof of Concept (PoC) has only been tested in a laboratory. The context of this PoC, its results and limitations will be discussed on this chapter.

In this chapter is first addressed the context of the PoC. Based on this context is then explained the scenario of the PoC. The code used by the as-Code tools is then explained. Based on this code is explained how to create and destroy the resources of the PoC. Finally, the results are observed and the limitations addressed.

5.1 Context

The global hardware context of the laboratory used for this PoC is the following;

- 1 × HP Proliant DL380 G7:
  12 CPUs hyper-thread 24,
  72GB RAM,
  2 × 10Gbps NIC,
  2 × 1To SATA SSD (RAID0),
  4 × 600GB SAS HDD (RAID5),
- 1 × HP Proliant DL380 G7:
  12 CPUs hyper-thread 24,
  72GB RAM,
  2 × 10Gbps NIC,
  8 × 300GB SAS HDD (RAID5),
- 1 × HP Proliant DL380 G7:
  6 CPUs hyper-thread 12,
  36GB RAM,
  2 × 10Gbps NIC,
  3 × 300GB SAS HDD (RAID5),
- 1 × Manageable switch:
  24 × 1 Gbps ports,
  4 × SFP+ ports,
- 1 × Physical firewall.

Schematically, the laboratory can be represented as following;

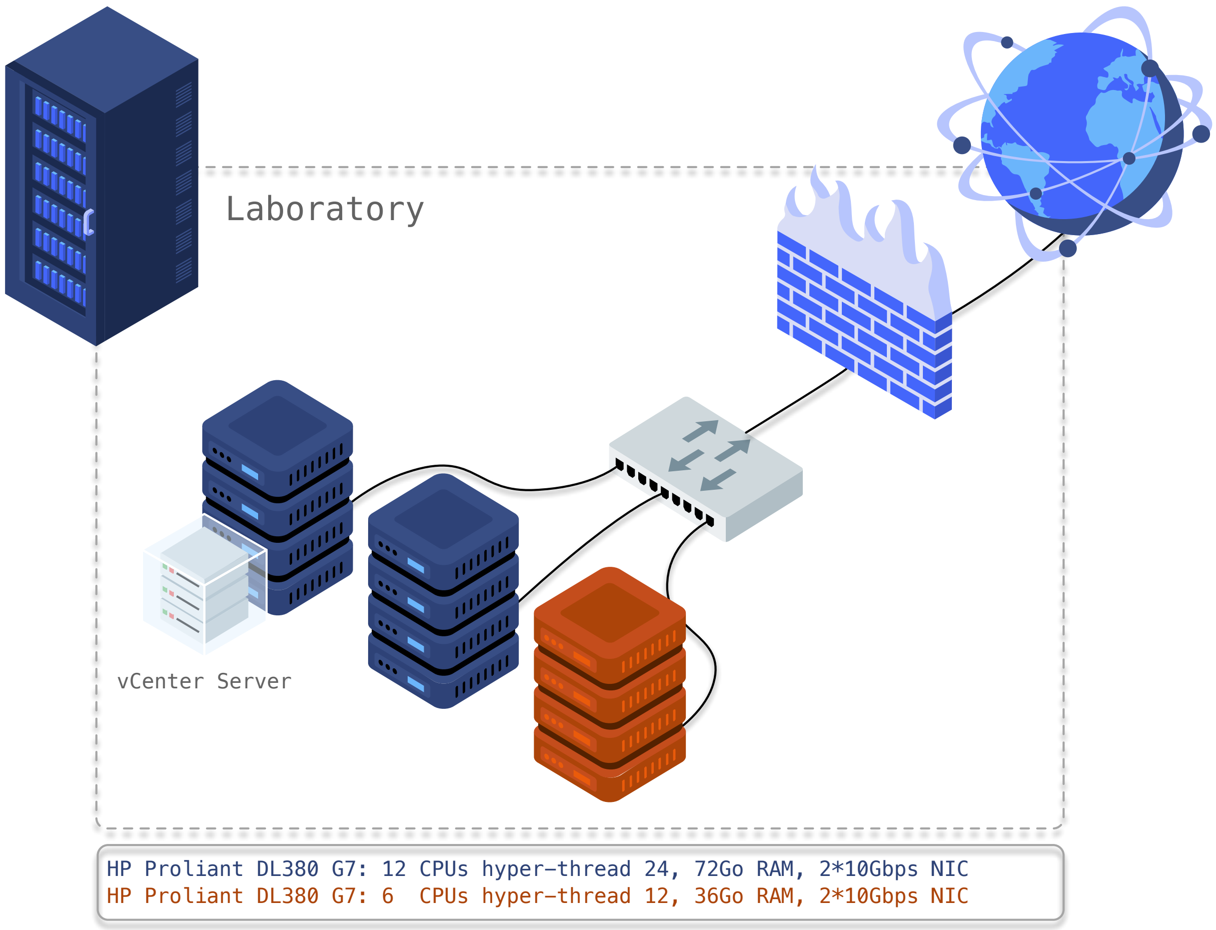

Laboratory
vCenter Server
HP Proliant DL380 G7: 12 CPUs hyper-thread 24, 72Go RAM, 2*10Gbps NIC
HP Proliant DL380 G7: 6 CPUs hyper-thread 12, 36Go RAM, 2*10Gbps NIC

Figure 5.1: Laboratory infrastructure.

5.1.1 Hypervisors

5.1.1.1 VMware ESXi

ESXi hypervisors used the base image: *HPE-ESXi-6.7.0-Update3-iso-Gen9plus-670.U3.10.4.5.19*[1]. ESXis were updated to the last available version, the last update is, *6.7.0, 16075168*. ESXi hypervisors are installed on HP Prolaint DL380 G7 with 12 CPUs (represented in blue of Figure 5.1).

5.1.1.2 VMware vCenter Server

The last update of the vCenter server is: *VMware vCenter Server Appliance 6.7.0.44000 Build number 16046470*.

The installation of ESXi hypervisors and the vCenter is anterior to the realization of this work.

5.1.1.3 KVM

The server with 6 CPUs (represented in orange in the Figure 5.1) is running Ubuntu 18.04.4 LTS with KVM version *2.11.1(Debian 1:2.11+dfsg-1ubuntu7.23)*. The use of KVM hypervisors aims to prove the compatibility of the concept with non-VMware host hypervisors.

[1]The *HPE-ESXi-6.7.0-Update3-iso-Gen9plus-670.U3.10.4.5.19* base image is supposed to run on the ninth generation (G9) of HP servers. This version of ESXi is not officially supported on the seventh generation (G7) of HP servers but runs without any issue.

5.1.1.4 Nested virtualization

VMware does not officially support nested virtualization. SLA and support are not assured when using nested virtualization. However, even if it is not officially supported by VMware, nested virtualization of ESXi works.[48] The additional layer of virtualization may impact performance and is the reason why VMware does not support it officially. If performance delta between standard ESXi usage and nested usage is relevant, dedicated hardware should be considered.

Nested ESXis are used on each of the three hypervisors of the laboratory. The installation was done with the same logic of installing standard ESXis; installed then upgraded to the last version: *6.7.0, 16075168.*

The nested virtual context is the following:

- On HP Proliant DL 380 G7 running ESXi, nested ESXis have;
  12 vCPUs,
  36GB RAM,
- On HP Proliant DL 380 G7 running KVM, nested ESXi has;
  8 vCPUs,
  24GB RAM,

Schematically, the nested infrastructure is the following;

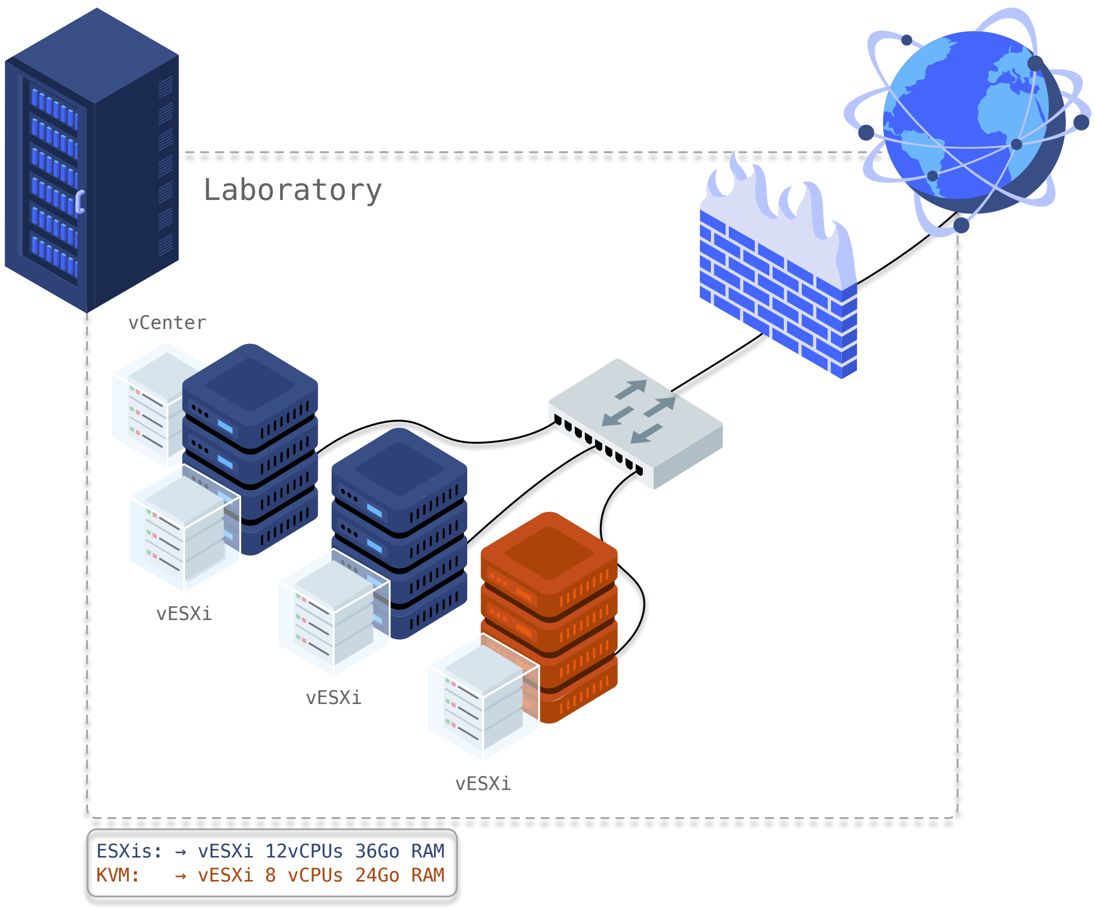

Laboratory
vCenter
vESXi
vESXi
vESXi
ESXis: → vESXi 12vCPUs 36Go RAM
KVM: → vESXi 8 vCPUs 24Go RAM

Figure 5.2: Laboratory nested infrastructure.

5.2 Scenario

The global scenario is to spawn and configure machines on nested hypervisors connected via VPN connections. Nested hypervisors are named;

- vesxi-u-01.rack,
- vesxi-u-02.rack,
- vesxi-r-03.rack.

Three sites are used to avoid the trivial case of only having two sites connected. The protocol used is OpenVPN. It creates L3 tunnels. IPSec could also have been used as easily as OpenVPN.

The VPN connections are performed by pfSense firewalls. Each hypervisor has one nested hypervisor and one pfSense VM running inside this nested hypervisor. They can be represented as following;

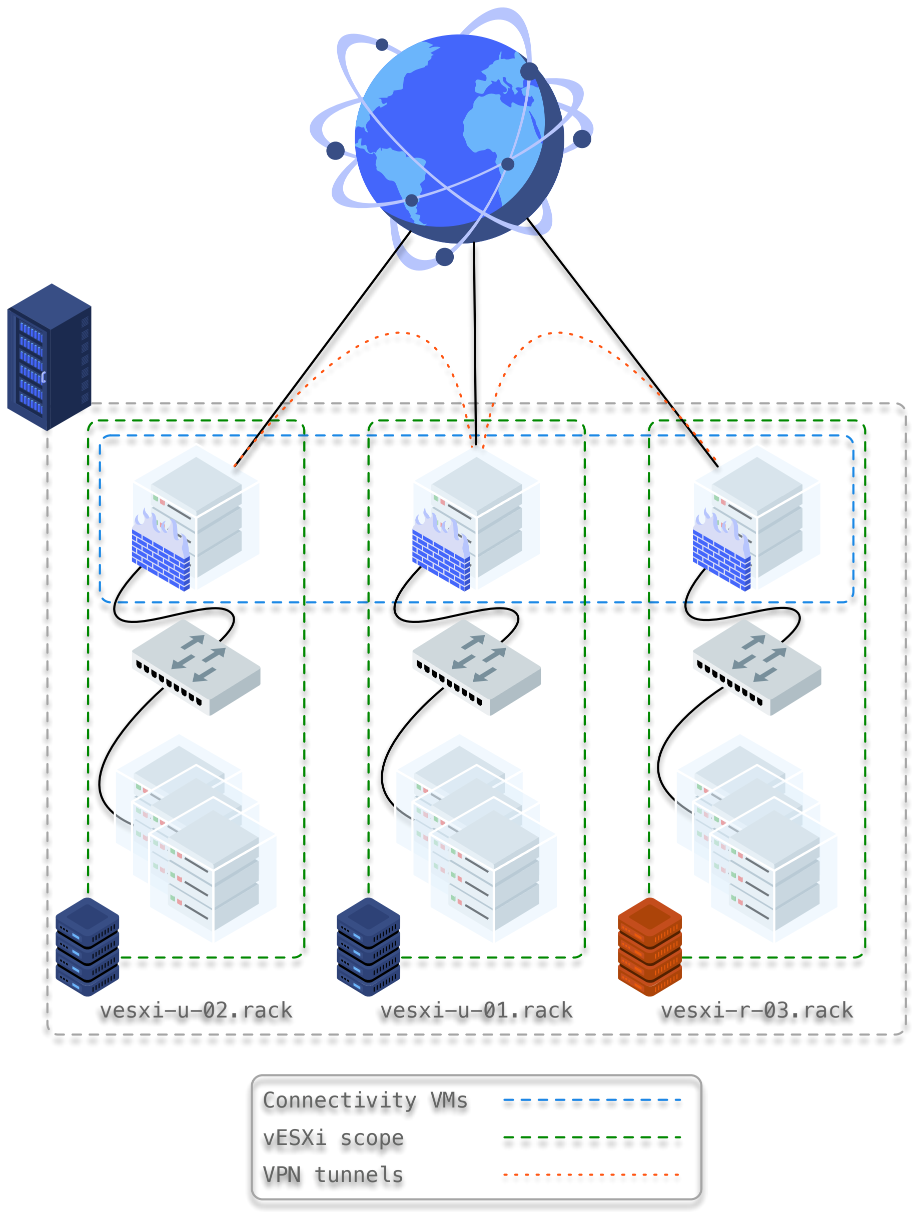


Figure 5.3: Laboratory nested infrastructure with connectivity.

The PoC scenario only has two layers of VMs connected to each other; the connectivity layer composed of pfSense firewalls and the machines behind the connectivity layer. It is, however, possible to add as much layer as required with the same logic. In this scenario, VMs connected to the connectivity layer are Ubuntu 18.04.04 LTS servers.

5.3 *as-Code* tools

Every machine, except ESXis, is built, deployed and configured via *as-Code* techniques. First, VMware VM templates are built with Packer. They are then deployed with Terrafom and configured with Ansible when required as shown in Figure 4.4. Packer builds VM templates on manual request. Indeed, there is no need to rebuild each template if there are no changes on the content of the template.

5.3.1 Packer

In this PoC Packer version 1.5.4 is used.

The VMware vCenter OS customization feature is used by Terraform, but does not support FreeBSD, on which pSfense is based. pfSense does not have built-in API capabilities. The configuration file can be injected in pfSense but the required packages need to be installed independently before injecting the configuration. To avoid such configuration injection and loss of time at boot, a dedicated VM template per pfSense is used. As shown on Figure 5.3, one pfSense instance is used on each vESXi to ensure VPN connections.

Before taking a look at the code, the Packer directory structure is represented as following:

```
packer
└── ubuntu-18.04
    ├── preseed.cfg
    ├── ubuntu-18.json
    └── variables.json
```

Listing 5.1: Packer file tree

The Listing 5.1 gives a simple representation of the file structure required to build the VM templates with Packer. The code required for the pfSense VM templates is similar except for the preseed file. For pfSense, boot commands and configuration file injection are used.

The code required to build the Ubuntu 18.04.04 LTS servers VM template is the following;

5.3.1.1 Variables

The following listing is the content of the *variables.json* file:

```
  {
    "vcenter_server": "vcenterserver.rack",
    "vsphere-datacenter": "FCR",
    "username": "{{env `PACKER_USERNAME`}}",
    "password": "{{env `PACKER_PASSWORD`}}",
    "datastore": "vESXI-U-01-DS1",
    "host": "vesxi-u-01.rack",
    "cluster": "vesxi-u-01.rack",
    "network": "VM Network - vesxi-u-01",
    "network-second": "VM Network - vesxi-u-01",
    "ssh_username": "ubuntu",
    "ssh_password": "ubuntu",
    "iso_url": "[vESXI-U-01-DS1] ISO/ubuntu-18.04.4-server-amd64.iso",
    "template_name": "Packer-Ubuntu-18.04-TEMPLATE-FCR"
  }
```

Listing 5.2: Packer Ubuntu server variables

Variables defined on Listing 5.2 are used to define the following parameters:

- The address of the vCenter server (line 2),
- The data center to use on the vCenter server (line 3),
- The credentials to connect to the vCenter server (lines 4-5),
- The datastrore to use on the destination ESXi (line 6),
- The destination ESXi on which the template will be built (line 7),
- The destination cluster (here the same than the ESXi) on which the template will be built (line 8),
- The virtual network connections (lines 9-10),
- The SSH credentials of the template (to run script on the template) (lines 11-12),
- The path to the ISO disk image on the destination ESXi (line 13),
- The name of the template to create (line 14).

5.3.1.2 Preseed

Preseed files are used to automate the deployment of Ubuntu machines. Those files are not Packer-specific but can still be used by Packer.[49] The following listing is the content of the *preseed.cfg* file:

```
1   d-i passwd/user-fullname string ubuntu
2   d-i passwd/username string ubuntu
3   d-i passwd/user-password password ubuntu
4   d-i passwd/user-password-again password ubuntu
5   d-i user-setup/allow-password-weak boolean true
6
7   d-i partman-auto/disk string /dev/sda
8   d-i partman-auto/method string regular
9   d-i partman-partitioning/confirm_write_new_label boolean true
10  d-i partman/choose_partition select finish
11  d-i partman/confirm boolean true
12  d-i partman/confirm_nooverwrite boolean true
13
14  d-i passwd/root-login boolean true
15  d-i passwd/root-password password ubuntu
16  d-i passwd/root-password-again password ubuntu
17
18
19  d-i pkgsel/include string open-vm-tools openssh-server cloud-init perl git ansible
20
21  d-i grub-installer/only_debian boolean true
22
23  d-i preseed/late_command string \
24      echo "@reboot root ping 8.8.8.8" >> /target//etc/crontab; \
25      echo ’ubuntu ALL=(ALL) NOPASSWD: ALL’ > /target/etc/sudoers.d/ubuntu ; \
26      in-target chmod 440 /etc/sudoers.d/ubuntu ;
27
28  d-i finish-install/reboot_in_progress note
```

Listing 5.3: Packer Ubuntu server preseed

The parameters defined on the Listing 5.3 are the following:

- The normal user credentials (lines 1-5),
- The disk partitioning (lines 7-12),
- The root user credentials (lines 14-16),
- The packages installation (line 19),
- The bootloader installation (line 21),
- The custom configuration part (lines 23-26),
- The end of the installation process (line 28),

5.3.1.3 Template

The template file is the file to provide to Packer to start the build process. This template uses *user variables*. These variables were defined on the variable file on subsubsection 5.3.1.1. User variables are used with the structure "{{ user ‘<varibale_name>‘ }}" where *<variable_name>* has to be replaced with the name of the variable to use *e.g.* "{{ user ‘password‘ }}".[50] The following listing is the content of the *ubuntu-18.json* file:

```
{
  "builders": [
    {
      "type": "vsphere-iso",

      "vcenter_server":      "{{user `vcenter_server`}}",
      "username":            "{{user `username`}}",
      "password":            "{{user `password`}}",
      "insecure_connection": "true",

      "datacenter": "{{user `vsphere-datacenter`}}",

      "vm_name": "{{user `template_name`}}",
      "datastore": "{{user `datastore`}}",
      "host":     "{{user `host`}}",
      "convert_to_template": "true",
      "cluster": "{{user `cluster`}}",
      "network": "{{user `network`}}",
      "boot_order": "disk,cdrom",

      "guest_os_type": "ubuntu64Guest",

      "ssh_username": "{{user `ssh_username`}}",
      "ssh_password": "{{user `ssh_password`}}",

      "CPUs":             4,
      "RAM":              4096,
      "RAM_reserve_all": false,

      "disk_controller_type":  "pvscsi",
      "disk_size":        15000,
      "disk_thin_provisioned": true,

      "network_card": "vmxnet3",

      "iso_paths": ["{{user `iso_url`}}"],
      "iso_url": "http://cdimage.ubuntu.com/releases/18.04/release/ubuntu-18.04.4-server-
          amd64.iso",
      "iso_checksum": "d5bc5c59c24191bb45dd85fc6a420b34",
      "iso_checksum_type": "md5",

      "floppy_files": [
        "./preseed.cfg"
      ],
      "boot_command": [
        "<enter><wait><f6><wait><esc><wait>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs><bs><bs><bs><bs><bs><bs><bs>",
        "<bs><bs><bs>",
        "/install/vmlinuz",
        " initrd=/install/initrd.gz",
        " priority=critical",
        " locale=en_US",
        " file=/media/preseed.cfg",
        "<enter>"
      ]
    }
  ],
  "provisioners": [
    {
      "type": "shell",
67         "inline": ["echo 'template build complete'"]
68       }
69     ]
70   }
```

Listing 5.4: Packer Ubuntu server template

The parameters defined on the Listing 5.4 are the following:

- The builder section; where the build part resides (lines 2-63),
  - The type of builder to use (line 4),
  - The vCenter connection with credentials using variables (line 6-9),
  - The data center selection on the vCenter using the *vsphere-datacenter* variable (line 11),
  - The VM name using the *template_name* variable (line 13),
  - The datastore name using the *datastore* variable (line 14),
  - The destination ESXi name using the *host* variable (line 15),
  - The template flag; flagging this VM as template when build is finished (line 16),
  - The destination cluster name using the *cluster* variable (line 17),
  - The network name using the *cluster* variable (line 18),
  - The boot order of the machine; disk then cdrom (line 19),
  - The OS type (line 21),
  - The SSH credentials (lines 23-24),
  - The resources allocated to the VM template (lines 26-32),
  - The network card type (line 34),
  - The ISO configuration (lines 36-39),
  - The preseed file configuration (lines 41-43),
  - The boot commands loading the preseed file (lines 44-61).
- The provisioner section; where the configuration part resides (lines 64-69),
  - The final *echo* signaling the end of the build.

5.3.1.4 Run

The Packer build can be started with the following bash command:

```
packer build -var-file variables.json ubuntu-18.json
```

Listing 5.5: Run Packer Ubuntu server template build

5.3.2 Terraform

In this PoC Terraform version 0.12 is used. Terrafom uses HCL, a JSON compatible file format.[51]

pfSense VMs are deployed in the PoC with Terraform as it is possible to reach each vESXi before the VPNs are deployed in the laboratory context. Outside of laboratory context, it would not be possible to deploy every VPN VM from a single point because they will require connectivity, that is set up by those exact VMs.

Doing such pfSense deployment in the laboratory only simplifies the context and proves the ability to deploy pfSense VMs without having to add another layer of VMs.

Terraform scripts are segmented in two parts;

1. The first is the previously evoked pfSense deployment script,
2. The second is deploying Ubuntu server VMs.

Before taking a look at the code, the (essential part of the) Terraform directory structure is represented as following:

```
1  tf-fcr
2  ├── 00-global-variables.tf
3  ├── 01-variables-u-01.tf
4  ├── 02-variables-u-02.tf
5  ├── 03-variables-r-03.tf
6  ├── 010-data-retrieve.tf
7  ├── 020-network.tf
8  ├── 060-ubuntu_vm.tf
9  ├── ansible
10 │   ├── apache2
11 │   │   ├── files
12 │   │   │   ├── apache.conf.j2
13 │   │   │   └── index.html.j2
14 │   │   ├── inventory.yml
15 │   │   ├── playbook.yml
16 │   │   └── vars
17 │   │       └── default.yml
18 │   ├── users
19 │   │   ├── playbook.yml
20 │   │   └── vars
21 │   │       └── default.yml
22 │   └── zsh
23 │       ├── playbook.yml
24 │       └── vars
25 │           └── default.yml
26 ├── packer
27 │   └── ubuntu-18.04
28 │       ├── preseed.cfg
29 │       ├── ubuntu-18.json
30 │       └── variables.json
31 ├── scripts
32 │   └── dns-ssh.sh
33 ├── terraform.tfstate
34 ├── terraform.tfstate.backup
35 └── versions.tf
```

Listing 5.6: Terraform file tree

The required structure for Terraform deployment is displayed in Listing 5.6. The Packer (lines 26-30) and Ansible (lines 9-25) directories are part of the Terraform directory and therefore are included on the listing. As for Packer listing, this one gives a simple representation of the file structure required to deploy resources with Terraform. The code structure required to deploy the pfSense VMs is similar to this one.

The content and purpose of each file or directory are the following:

- 00-global-variables.tf: definition of global variables used by multiple files,
- x-variables-y-z.t: definition of variables proper to specific resources,
- 010-data-retrieve.tf: retrieval of data on the vCenter,
- 020-network.tf: creation of networking resources,
- 060-ubuntu_vm.tf: creation of Ubuntu 18.04 servers,
- packer: Packer directory; as explained on subsection 5.3.1,
- ansible: Ansible directory; will be explained on subsection 5.3.3,
- scripts: script directory
  - dns-ssh.sh: script pushing DNS configuration and SSH public keys on created VM,
- terraform.tfstate: Terraform-specific file containing the state of the resources,
- terraform.tfstate.backup: Terraform-specific file containing a backup of the state of the resources,
- versions.tf: Terraform-specific file containing the version of Terraform used.

Essential content

The code shown from subsubsection 5.3.2.1 to subsubsection 5.3.2.5 is the essential extracted code required to deploy two Ubuntu 18.04.04 LTS server VMs on only one host; *vesxi-u-01*;

5.3.2.1 Global variables

Global variables file is used to define variables related to the vCenter and establish its connection. Varibles common to the entire deployment are also defined is this file. The following listing is the essential content (as defined in section 5.3.2) of the *00-global-variables.tf* file:

```
variable "vcenter_user" {
  default = "user"
}

variable "vcenter_password" {
  default = "password"
}

variable "vcenter_vsphere_server" {
  default = "vcenterserver.rack"
}

provider "vsphere" {
  user           = var.vcenter_user
  password       = var.vcenter_password
  vsphere_server = var.vcenter_vsphere_server
  allow_unverified_ssl = true
}

variable "template_image_ubuntu_18_04" {
  default = "Packer-Ubuntu-18.04-TEMPLATE-FCR"
}

variable "dc" {
  default = "FCR"
}

variable "hosts" {
  default = [
    {
      name = "vesxi-u-01"
      hostname = "vesxi-u-01.rack"
    }
  ]
}

variable "clusters" {
  type = map
  default = {
    vesxi-u-01 = "vesxi-u-01.rack"
  }
}

variable "vswitches" {
  type = map
  default = {
    vesxi-u-01-vs-0 = "vSwitch0"
    vesxi-u-01-vs-1 = "vSwitch1"
  }
}

variable "port_groups" {
  type = map
  default = {
    vesxi-u-01-pg-0 = "VM Network - vesxi-u-01"
  }
}
```

Listing 5.7: Terraform global variables

The variables defined on the Listing 5.7 are the following:

- The credentials of the vCenter (lines 1-7),
- The address of the vCenter (lines 9-11),
- The connection to the vCenter: with *allow_unverified_ssl* set to true to allow the use of the self-signed ssl certificates (lines 13-19),
- The name of the template to use for the Ubuntu 18.04 machines on the vCenter (lines 20-22),
- The name of the data center to use for the vCenter (lines 24-26),
- The host(s) names (ESXi(s)) definition (lines 28-35),
- The cluster(s) name (ESXi(s) in this case) definition (lines 37-42),
- The virtual switche(s) name definition (lines 44-50),
- The port group(s) name definition (lines 52-57),

5.3.2.2 Variables for Virtual Machine(s)

The following listing is the essential content (as defined in section 5.3.2) of the *01-variables-u-01.tf* file:

```
1   variable "ubuntu_vm_params_vesxi-u-01" {
2     default = {
3       vcpu = "4"
4       ram  = "4096"
5       disk_datastore = "vESXI-U-01-DS1"
6       disk_size      = "25"
7     }
8   }
9
10  variable "ubuntu_network_params_vesxi-u-01" {
11    default = {
12      domain        = "test.local"
13      label         = "ubuntu_network_vesxi-u-01"
14      vlan_id       = "0"
15      subnet        = "172.10.0.0/24"
16      gateway       = "172.10.0.254"
17      dns           = ["8.8.8.8", "8.8.4.4"]
18    }
19  }
20
21  variable "ubuntu_base_hostname_vesxi-u-01" {
22    default = "u-01-ubuntu0"
23  }
24
25  variable "ubuntu_vm_desired_capacity_vesxi-u-01" {
26    default = "2"
27  }
```

Listing 5.8: Terraform variables for Virtual Machine(s)

The variables defined on the Listing 5.8 are the following:

- The variables defining resources allocated to the VM(s): number of virtual CPU(s), amout of RAM, disk space and destination datastore (lines 1-8),
- The domain used on the deployed machines lines (line 12),
- The label (the name) fo the port group (line 13),
- The VLAN id of the port group (line 14),
- The network configuration of the machine (subnet, gateway, DNS) (lines 15-17)
- The base name of the machines deployed (lines 21-23),
- The number of machines to deploy (lines 25-27).

5.3.2.3 Data retrieval

Data retrieval is required to have the object (hosts, datastores, port groups, etc.) IDs from the vCenter. The following listing is the essential content (as defined in section 5.3.2) of the

010-data-retrieve.tf file:

```
 1  data "vsphere_datacenter" "dc" {
 2    name = var.dc
 3  }
 4
 5  data "vsphere_host" "vesxi-u-01" {
 6    name          = var.clusters.vesxi-u-01
 7    datacenter_id = data.vsphere_datacenter.dc.id
 8  }
 9
10  data "vsphere_resource_pool" "vesxi-u-01" {
11    # If you haven't resource pool, put "Resources" after cluster name
12    name          = "${var.clusters.vesxi-u-01}/Resources"
13    datacenter_id = data.vsphere_datacenter.dc.id
14  }
15
16  data "vsphere_datastore" "ubuntu-vesxi-u-01" {
17    name          = var.ubuntu_vm_params_vesxi-u-01["disk_datastore"]
18    datacenter_id = data.vsphere_datacenter.dc.id
19  }
20
21  data "vsphere_network" "ubuntu-vesxi-u-01" {
22    name          = var.ubuntu_network_params_vesxi-u-01["label"]
23    datacenter_id = data.vsphere_datacenter.dc.id
24    depends_on    = [vsphere_host_port_group.ubuntu_port-vesxi-u-01]
25  }
26
27  data "vsphere_virtual_machine" "template_ubuntu_18_04" {
28    name          = var.template_image_ubuntu_18_04
29    datacenter_id = data.vsphere_datacenter.dc.id
30  }
```

Listing 5.9: Terraform data retrieval

The elements retrieved from the Listing 5.9 are the following:

- The data center (lines 1-3),
- The host (ESXi) (lines 5-8),
- The resource pool (lines 10-14),
- The datastore (lines 16-19),
- The network (lines 21-25),
- The virtual machine template (lines 27-30).

5.3.2.4 Port group creation on vSwitch

The following listing is the essential content (as defined in section 5.3.2) of the *020-network.tf* file:

```
  resource "vsphere_host_port_group" "ubuntu_port-vesxi-u-01" {
    name                = var.ubuntu_network_params_vesxi-u-01["label"]
    host_system_id      = data.vsphere_host.vesxi-u-01.id
    virtual_switch_name = var.vswitches.vesxi-u-01-vs-1
    vlan_id             = var.ubuntu_network_params_vesxi-u-01["vlan_id"]
    allow_promiscuous   = true
}
```

Listing 5.10: Terraform port group creation on vSwitch

The result on the vCenter of such configuration is the following (highlited by the blue rectangle):

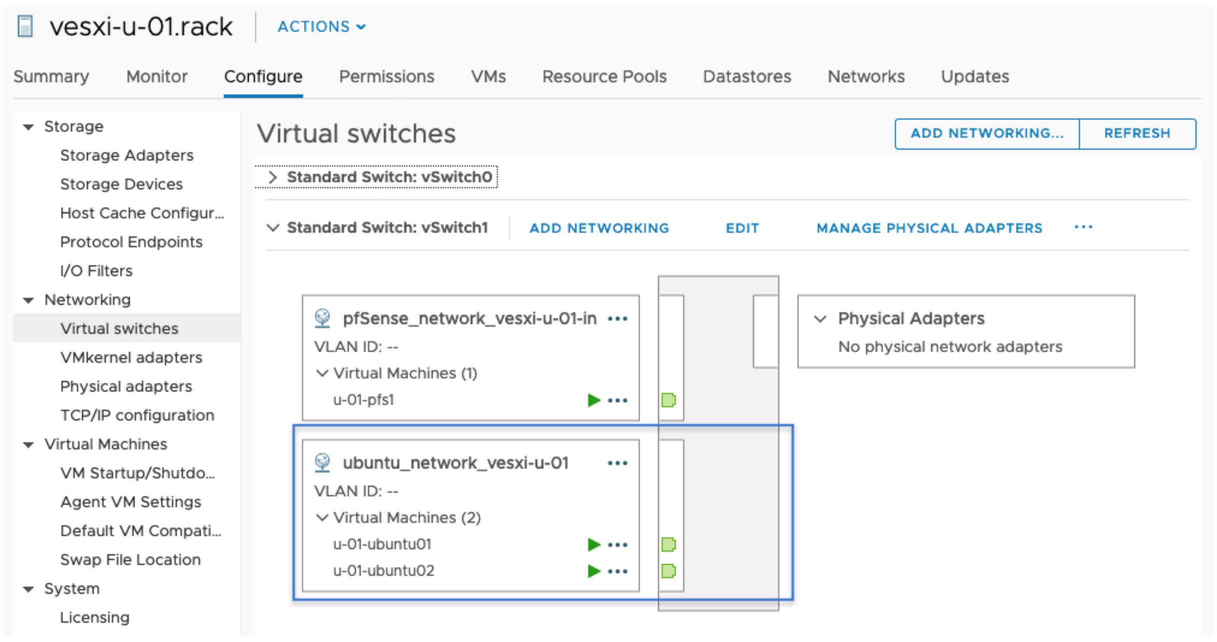


Figure 5.4: Results of port group creation.

5.3.2.5 Virtual Machine(s) creation

This part of the code is managing the creation of the VM(s) and running Ansible to configure them. The following listing is the essential content (as defined in section 5.3.2) of the *060-ubuntu_vm.tf* file:[2]

```
# Create random passwords
resource "random_password" "password-u-01" {
  count = var.ubuntu_vm_desired_capacity_vesxi-u-01
  length = 32
  special = false
}

# Create the VM(s)
resource "vsphere_virtual_machine" "ubuntu_vm_vesxi-u-01" {
  count            = var.ubuntu_vm_desired_capacity_vesxi-u-01
  name             = "${var.ubuntu_base_hostname_vesxi-u-01}${count.index + 1}"
  num_cpus         = var.ubuntu_vm_params_vesxi-u-01["vcpu"]
  memory           = var.ubuntu_vm_params_vesxi-u-01["ram"]
  datastore_id     = data.vsphere_datastore.ubuntu-vesxi-u-01.id
  host_system_id   = data.vsphere_host.vesxi-u-01.id
  resource_pool_id = data.vsphere_resource_pool.vesxi-u-01.id
  guest_id         = data.vsphere_virtual_machine.template_ubuntu_18_04.guest_id
  scsi_type        = data.vsphere_virtual_machine.template_ubuntu_18_04.scsi_type
  annotation       = "ubuntu:ubuntu"

  # Define network interface
  network_interface {
    network_id = data.vsphere_network.ubuntu-vesxi-u-01.id
  }

  # Define disk parameters
  disk {
    name = "${var.ubuntu_base_hostname_vesxi-u-01}${count.index + 1}.vmdk"
    size = var.ubuntu_vm_params_vesxi-u-01["disk_size"]
  }

  # Define template and customisation parameters
  clone {
```

[2]For clarity reasons, the explanations are given by the comments on the code (comments are identified by a line starting with #)

```
    template_uuid = data.vsphere_virtual_machine.template_ubuntu_18_04.id

    customize {
      linux_options {
        host_name = "${var.ubuntu_base_hostname_vesxi-u-01}${count.index + 1}"
        domain    = var.ubuntu_network_params_vesxi-u-01["domain"]
      }

      network_interface {
        ipv4_address    = cidrhost(var.ubuntu_network_params_vesxi-u-01["subnet"], count.index + 10)
        ipv4_netmask    = split("/", var.ubuntu_network_params_vesxi-u-01["subnet"])[1]
        dns_server_list = var.ubuntu_network_params_vesxi-u-01["dns"]
      }

      ipv4_gateway = var.ubuntu_network_params_vesxi-u-01["gateway"]
    }
  }

  # Define dependency on the port group on the vswitch
  depends_on = [vsphere_host_port_group.ubuntu_port-vesxi-u-01]

  # Inject base script file injecting RSA SSH key and fixing DNS
  provisioner "file" {
    connection {
      type     = "ssh"
      user     = "ubuntu"
      password = "ubuntu"
      host     = cidrhost(var.ubuntu_network_params_vesxi-u-01["subnet"], count.index + 10)
    }
    source      = "scripts/dns-ssh.sh"
    destination = "/tmp/dns-ssh.sh"
  }

  # Running the script on the remote machine
  provisioner "remote-exec" {
    connection {
      type     = "ssh"
      user     = "ubuntu"
      password = "ubuntu"
      host     = cidrhost(var.ubuntu_network_params_vesxi-u-01["subnet"], count.index + 10)
    }
    inline = [
      "chmod +x /tmp/dns-ssh.sh",
      "sudo /tmp/dns-ssh.sh",
    ]
  }

  # Run ansible playbook with the previously injected RSA SSH key; installing custom shell
  provisioner "local-exec" {
    command = "ansible-playbook ansible/zsh/playbook.yml -u ubuntu -i ${cidrhost(var.ubuntu_network_params_vesxi-u-01["subnet"], count.index + 10)},"
  }

  # Run ansible playbook with the previously injected RSA SSH key; installing apache2 with configuration
  provisioner "local-exec" {
    command = "ansible-playbook ansible/apache2/playbook.yml -u ubuntu -i ${cidrhost(var.ubuntu_network_params_vesxi-u-01["subnet"], count.index + 10)}, -e 'http_host=${var.ubuntu_base_hostname_vesxi-u-01}${count.index + 1}'"
  }

  # Run ansible playbook with the previously injected RSA SSH key; change credentials of the machine
  provisioner "local-exec" {
    command = "ansible-playbook ansible/users/playbook.yml -u ubuntu -i ${cidrhost(var.ubuntu_network_params_vesxi-u-01["subnet"], count.index + 10)}, -e 'password=${random_password.password-u-01[count.index].result}'"
  }
}
```

```
# Display IP addresses of the machines created by Terraform
output "Ubuntu-u-01-IPs" {
  value       = zipmap(vsphere_virtual_machine.ubuntu_vm_vesxi-u-01[*].name,
      vsphere_virtual_machine.ubuntu_vm_vesxi-u-01[*].default_ip_address)
  description = "The IP addresses of all ubuntu machines on u-01"
}

# Display passwords of the machines created by Terraform
output "Ubuntu-u-01-Passwords" {
  value       = try(zipmap(vsphere_virtual_machine.ubuntu_vm_vesxi-u-01[*].name,
      random_password.password-u-01[*].result), "None")
  description = "The passwords of all ubuntu machines on u-01"
}
```

Listing 5.11: Terraform Virtual Machine(s) creation

5.3.3 Ansible

In this PoC Ansible version 2.9 is used.

Also in this PoC, Ansible runs three playbooks, their usage is the following;

- Install custom shells; this playbook illustrates were custom scripts can be integrated into Ansible,
- Install Apache2 with basic configuration (playbook from GitHub of Digital Ocean community [52]); this playbook illustrates standard Ansible playbook capabilities,
- Credentials update of the machine; this playbook updates the default credential of the VM template for security reasons.

Those playbooks are launched on Ubuntu server VMs. The following playbook is used to update passwords based on the Terraform *random_password* resource;

5.3.3.1 Credentials update

```
---
- hosts: all
  become: true
  vars_files:
    - vars/default.yml

  tasks:
    - name: Set ubuntu password
      user:
        name: ubuntu
        password: "{{ password | password_hash('sha512') }}"
    - name: Set root password
      user:
        name: root
        password: "{{ password | password_hash('sha512') }}"
```

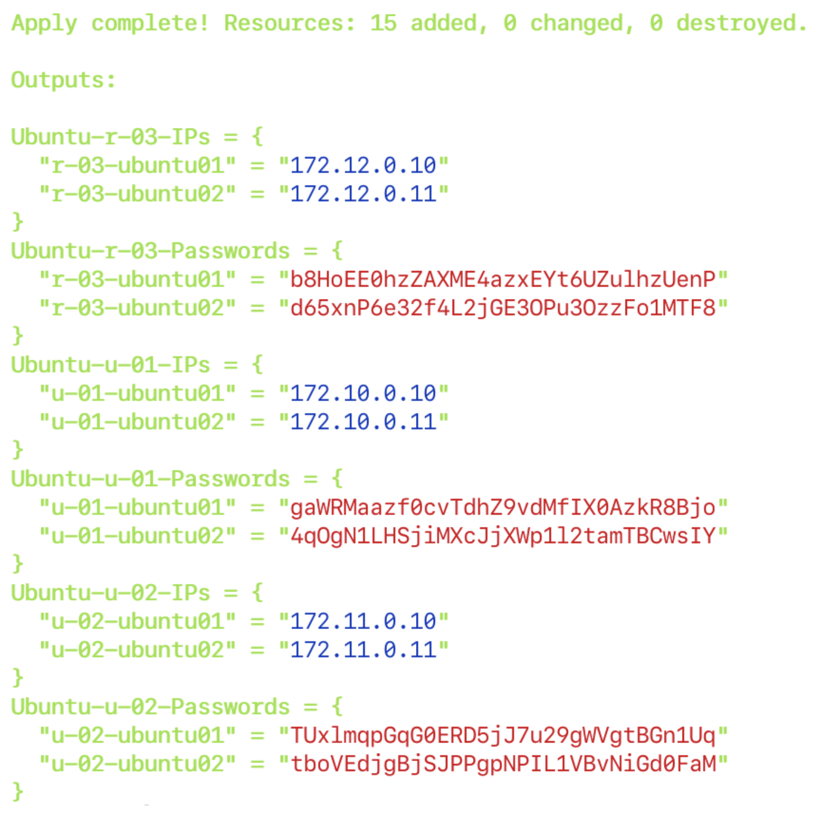
Listing 5.12: Ansible update users credentials

Such short playbook only aims to show that any standard Ansible playbook can be used by Terrafrom.

5.4 Run

To run the Terrafrom script, it is only required to perform a *terraform apply* bash command within the directory containing the Terrafom code. Terraform will then create the resources declared in the code.

The final output of Terraform should be similar to the following screenshot;



```
Apply complete! Resources: 15 added, 0 changed, 0 destroyed.

Outputs:

Ubuntu-r-03-IPs = {
  "r-03-ubuntu01" = "172.12.0.10"
  "r-03-ubuntu02" = "172.12.0.11"
}
Ubuntu-r-03-Passwords = {
  "r-03-ubuntu01" = "b8HoEE0hzZAXME4azxEYt6UZulhzUenP"
  "r-03-ubuntu02" = "d65xnP6e32f4L2jGE3OPu3OzzFo1MTF8"
}
Ubuntu-u-01-IPs = {
  "u-01-ubuntu01" = "172.10.0.10"
  "u-01-ubuntu02" = "172.10.0.11"
}
Ubuntu-u-01-Passwords = {
  "u-01-ubuntu01" = "gaWRMaazf0cvTdhZ9vdMfIX0AzkR8Bjo"
  "u-01-ubuntu02" = "4qOgN1LHSjiMXcJjXWp1l2tamTBCwsIY"
}
Ubuntu-u-02-IPs = {
  "u-02-ubuntu01" = "172.11.0.10"
  "u-02-ubuntu02" = "172.11.0.11"
}
Ubuntu-u-02-Passwords = {
  "u-02-ubuntu01" = "TUxlmqpGqG0ERD5jJ7u29gWVgtBGn1Uq"
  "u-02-ubuntu02" = "tboVEdjgBjSJPPgpNPIL1VBvNiGd0FaM"
}
```

Figure 5.5: Result of *terraform apply* bash command.

As password credentials are generated by Terraform, they will remain in the Terraform state and not be altered if some resources are added or destroyed.

5.5 Destroy

Terrafom is very powerful in terms of destruction of resources. Indeed to destroy the resources it is only required to perform a *terraform destroy* bash command within the directory containing the Terrafom code.

5.6 Results

5.6.1 Creation

The time to create resources is dependent on multiple factors. The hardware is the main bottleneck, and more specifically, the network connection between sites, the disks speed, the amount of resources created in parallel and in total, the size of the resources created, etc.

The following two tests and their results are designed to give an indication of the time required to deploy resources;

5.6.1.1 2 VMs on each vESXi

The deployed VMs have 4 vCPu, 4GB of RAM and 25GB of disk. The size on disk of machines deployed is 6,5GB. The VM template is located on the datastore of the vesxi-u-01.

The time dispatch is the following;

- Creation of network resources on all vESXis: ~ 0s,

- Clone on vesxi-u-01: ~ 2m,
- Clone on vesxi-u-01 with the 3 Ansible playbooks: ~ 3m49s,

- Clone on vesxi-u-02: ~ 2m30s,
- Clone on vesxi-u-02 with the 3 Ansible playbooks: ~ 4m10s,

- Clone on vesxi-r-03: ~ 5m,
- Clone on vesxi-r-03 with the 3 Ansible playbooks: ~ 7m10s,

Sum of 6 VMs created and configured in ~ 7m30s.

5.6.1.2 5 VMs on each vESXi

The deployed machines have 4 vCPu, 4GB of RAM and 25GB of disk. The size on disk of machines deployed if 6,5GB. The VM template is located on the datastore of the vesxi-u-01.

The time dispatch is the following;

- Creation of network resources on all vESXis: ~ 0s,

- Clone on vesxi-u-01: ~ 2m to 2m20s,
- Clone on vesxi-u-01 with the 3 Ansible playbooks: ~ 4m10s to 4m20s,

- Clone on vesxi-u-02: ~ 2m30s,
- Clone on vesxi-u-02 with the 3 Ansible playbooks: ~ 4m10s to 5m20s,

- Clone on vesxi-r-03: ~ 9m,
- Clone on vesxi-r-03 with the 3 Ansible playbooks: ~ 11m30s to 11m40s,

Sum of 15 VMs created and configured in around 20m.

The time delta between the two examples is due to the default setup of parallel task run by Terraform which is 10. More load was also going through the network between machines. A bottleneck can be due to the limitation of the vesxi-r-03 running under KVM hypervisor which is not able to map the 10Gbps NIC to the hypervisors. Indeed on the vesxi-r-03, the 10Gbps NIC appears as a 1Gbps NIC.

5.6.2 Destruction

The destruction is fast, a few seconds was enough to destroy the infrastructure in both previous evoked examples.

5.6.3 Benchmark

Benchmarking was performed to estimate the performance loss due to nested virtualization. The phoronix-test-suite[53] tool was used to conduct the benchmarks. The version 9.6.0 was used. Benchmarks were performed on VMs and nested VMs on host virtualizing the vesxi-u-01 and on the KVM host. VMs used for the benchmarks were Ubuntu 18.04 LTS server with 4vCPU, 4GB RAM and 25GB disk (thin provisioned).

5.6.3.1 I/O - Disk

To benchmark the I/O disk performances the *pts/iozone 1.9.6*[54] benchmark was used. The parameters used were the following:

- Record Size: 1MB,
- File Size: 4GB,
- Disk Test: Read and write (all options).

The configurations of the disk bus on the KVM host were the following:

- Ubuntu 18.04 standard VM: *VirtIO*,
- ESXi nested VM: *IDE*.

The ESXi nested VM does not use the *VirtIO* disk bus because disks were not detected on the nested ESXi VM when the *VirtIO* bus type was tested. The only configuration on which the disks were detected was *IDE*. The results are the following:

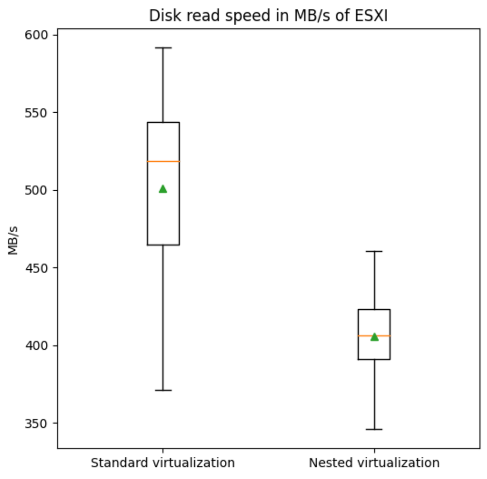


Figure 5.6: I/O read speed of ESXi system.

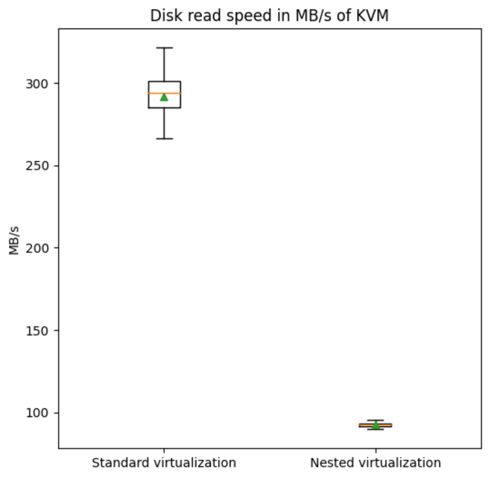


Figure 5.7: I/O read speed of KVM system.

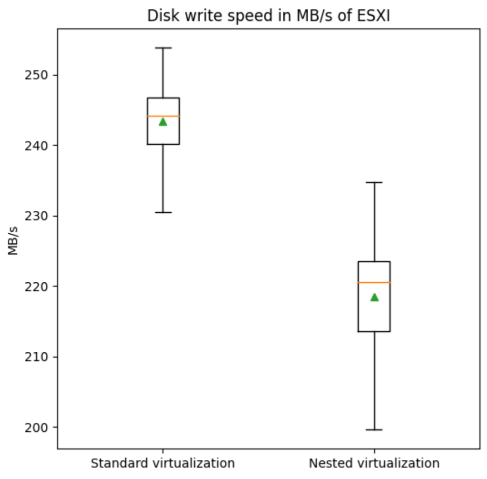


Figure 5.8: I/O write speed of ESXi system.

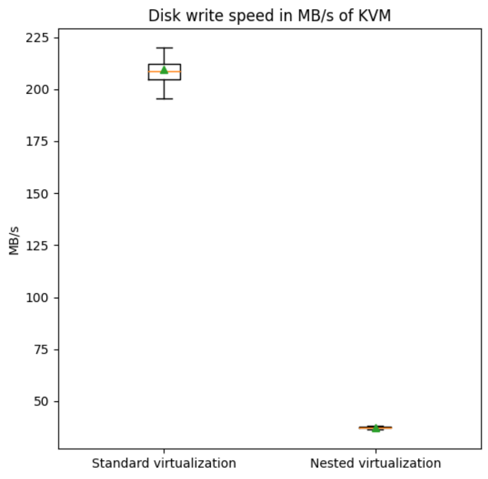


Figure 5.9: I/O write speed of KVM system.

The Figure 5.6, Figure 5.7, Figure 5.8 and Figure 5.9, standard and nested, are all based on 100 samples.

Observations

On the previous Figure 5.6, Figure 5.7, Figure 5.8 and Figure 5.9, the median is represented as the orange line and the mean is represented as the green triangle. Medians and means can be used to compare the results of the benchmarks.

Also on the same previous figures, performance loss is to be observed between standard and nested virtualization. Indeed, the mean and the median of nested virtualization performance are lower than the mean and median of standard virtualization. It is only relevant to compare the plot boxes of the same figures. Indeed, the aim of the figures is to compare standard and nested virtualization of machines running on the same physical hardware. The goal of such observation is not to compare the performance between different physical machine but only observe the performance loss (if any) introduced by the layers of virtualization.

Therefore, when comparing standard and nested virtualization of each figure individually, disk performance loss is to be observed. This performance loss is the result of additional virtualization layer.

It is worth mentioning that ESXis were not configured with cache disks. However, a delta between the benchmark result and the monitoring embedded inside the vCenter exists. The monitoring outputs can be found on the Appendix A and more precisely on Figure A.1. The maximum disk read value observed by the built-in monitoring is around 420MBps. The delta between the maximum value returned by the benchmark (around 600 MBps) and the by the built-in monitoring is probably the result of some disk caching at the ESXi or Ubuntu itself level.

5.6.3.2 CPU

To benchmark the CPU performances the *pts/compress-7zip 1.7.1* [55] benchmark was used. No parameter is required for this benchmark. The results are the following:

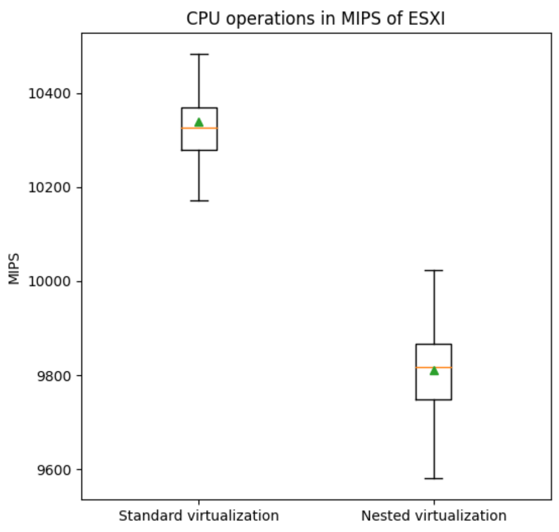


Figure 5.10: CPU speed of ESXi system.

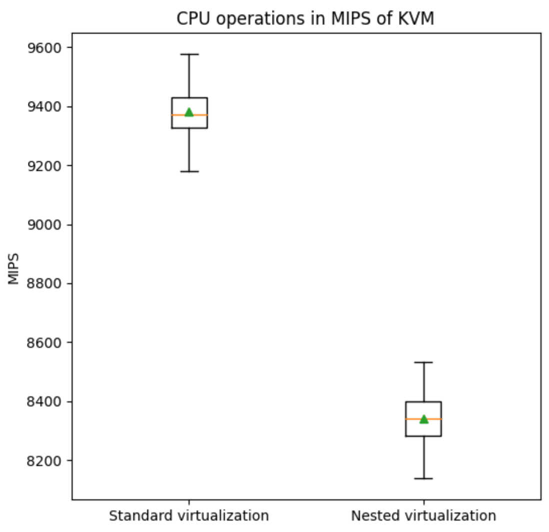


Figure 5.11: CPU speed of KVM system.

The Figure 5.10 and Figure 5.11, standard and nested, are all based on 100 samples. MIPS stands for Million Instructions Per Second.[56]

Observations

On the previous Figure 5.10, and Figure 5.11, the median is represented as the orange line and the mean is represented as the green triangle. Medians and means can be used to compare the results of the benchmarks.

The observation realized concerning disk performance loss can be applied for the CPU. Indeed, on the Figure 5.10 and Figure 5.11 a CPU performance loss is to be observed. The reason of this performance loss is identical to the disk performance loss *i.e.*, to the additional virtualization layer.

5.6.3.3 RAM

To benchmark the RAM performances the *pts/mbw 1.0.0*[57]
benchmark was used. The parameters used were the following:

- Test: Memory Copy,
- Array Size: 1024 MiB.

The results are the following:

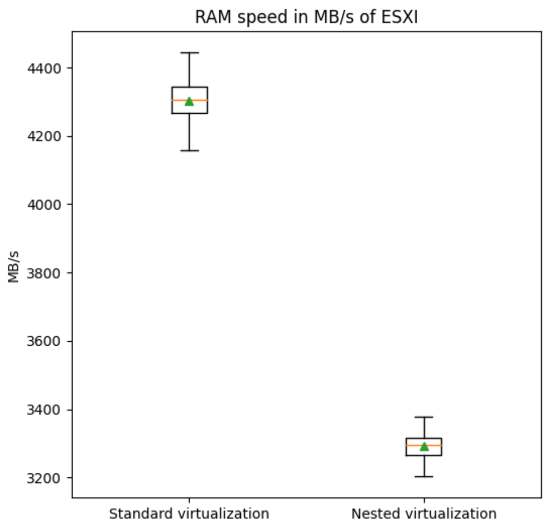


Figure 5.12: RAM speed of ESXi system.

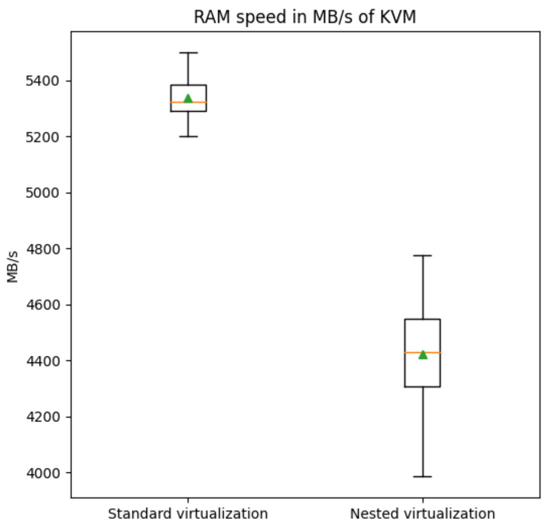


Figure 5.13: RAM speed of KVM system.

The Figure 5.12 and Figure 5.13, standard and nested, are all based on 100 samples.

Observations

On the previous Figure 5.12 and Figure 5.13, the median is represented as the orange line and the mean is represented as the green triangle. Medians and means can be used to compare the results of the benchmarks.

The observation realized concerning disk performance loss can be applied for the RAM. Indeed, on the Figure 5.12 and Figure 5.13 a RAM performance loss is to be observed. The reason of this performance loss is identical to the disk and CPU performance loss *i.e.*, to the additional virtualization layer.

5.7 Remarks

All the code displayed previously in this chapter is only a part of the code. The whole code is split in +50 files and can be found on GitHub:

- VPNs: `https://github.com/NicodemeB/tf-vpns`,
- FCR: `https://github.com/NicodemeB/tf-fcr`.

5.8 Limitations

During experimentations, build and different tests, limitations regarding the concept were found. They are the following:

5.8.1 VMware OS customization

The VMware OS customization feature used by Terraform, as stated in subsection 5.3.1, suffers from limitations. Indeed, not all OSs are supported by this feature.[58] VMware support in particular multiple version of Microsoft Windows, Red Hat, CentOS, Oracle Linux, Ubuntu and SLES. It should, however, be possible to bypass this limitation with Packer.

5.8.2 Nested virtualization

During different tests, the support of nested virtualization of ESXi on other hypervisors than ESXi and KVM was tested. VMware Fusion was rapidly tested and functional. VirtualBox was extensively tested. As a matter of fact, VirtualBox supports nested virtualization since version 6.0.[59] It was, however, impossible to make a vESXi with its additional Virtual Machines to run. All nested virtualization options were tested on multiple different hardware always obtaining the same negative result.

5.8.3 KVM

5.8.3.1 10G NIC

As already introduced in the subsection 5.6.1 10Gbps interface is bottlenecked to 1Gbps in nested ESXi on KVM. On KVM, multiple drivers are available, but only the *e1000* driver resulted in a working network adapter on the nested ESXi. Using such driver, the result is a 1Gbps NIC. Screenshots are available on Appendix B

5.8.3.2 Disks bus type

As already mentioned in the subsubsection 5.6.3.1 the *VirtIO* bus type was not working on the nested ESXi. When this type of bus was tested the result was that the disks were not detected by the nested ESXi. *IDE* was the only bus type working, which is less performant the *VirtIO* bus type but offers more compatibility.

5.8.3.3 Hypervisors and VMs performances

As seen in the subsection 5.6.3, variable performance loss is possible due to the additional layer of nested virtualization. This limitation and the resulting performance should be considered before using nested virtualization.

5.9 PoC conclusion

The elements shown and explained in this chapter demonstrated the concept exposed on chapter 4 which were automation, hypervisors support and connectivity. Indeed, automation between multiple site running on multiple hypervisors was demonstrated as working. Limitations regarding the concept and the PoC were also addressed.

6. Future work

This chapter aims to describe future work that could be the next step of this work.

In this chapter is first discussed the scenario abstraction. The automation is then evoked, followed by the router emulation and identity management and finally, the logging and monitoring.

6.1 Scenarios abstraction

The main future work is developing the abstraction mechanism in the scenarios. As a matter of fact, there are a lot of tasks that can be achieved with the concept of this work. However, this requires a Terraform, Packer and Ansible expertise to make it work. The approach about abstraction in scenarios could be to use a language (or create one if required) to structure scenarios. This piece of code could then be transformed in the appropriate IaC and CaC tool and then deploy and configure the required resources.

YAML or JSON language could be used to declare the template files. This will, however, still come with knowledge requirements for the declaration of such template file. The best alternative will probably be to design an application, probably a web application, capable of guiding users to generate templates then perform the resources deployment and configuration.

One of the drawbacks of such solution is once more, the limitations of the application itself. At some point it will be difficult to design complex topologies without having to deal with complex deployment challenges. Graphical representation can improve clarity but configuration will still have limitations in terms of non-expert users. It would probably be possible to create few template and derive configurations from them but at some point if custom configuration is required, custom code will follow.

6.2 Automation

6.2.1 Packer automation

Actually, in the actual form of the concept and the PoC the Packer builds of VM templates are launched manually. The next step could be to be able to determine when the build instructions have changed and integrated the build of required VM templates to Terraform. Doing such thing could also help reduce the time required for deployments if VM templates are not copied from a single ESXi but are distributed in the required destination ESXis. Storage space on each ESXi should be considered before doing such thing. Indeed the total space occupied by the VM templates will be higher if templates are copied on multiple ESXis.

The speed of deployments will probably take more time when VM templates will require to be built.

6.2.2 Hypervisor's deployment automation

Deployment of ESXis are actually performed manually. One of the next steps could be to create an image template of ESXi. This image could be used for deploying ESXis or virtual ESXis. Creating such template could improve the deployment speed when adding new hardware to the federation.

6.3 Router emulation

Emulation of routers such as Cisco IOS should be theoretically possible in the actual concept and PoC. It will, however, require lots of customization to make it work. Another next step could be to improve this process and integrate it to the actual concept. Configuration of such devices should also be addressed.

Improving such functionality will increase the range of supported devices and should improve realism.

6.4 Identity management

At the moment, the PoC is using flat role segmentation. When accessing machines, it is all or nothing, meaning all rights or no rights at all. When deploying large scenarios, this concern should be addressed to segment rights and access to machines as discussed in section 3.5. Indeed users and federation administrators should be distinguished. As also discussed in section 3.6, segmentation of users right restricting access to certain resources should also be addressed.

At the moment there is also no directory integration nor central AAA system. Adding those elements will probably help with management of credentials.

6.5 Logging & monitoring

As discussed in section 3.7, logging and monitoring could be useful in certain scenarios. This layer could be centralized and collect logs from resources deployed. Those deployed resources could be configured to use the central system as logging server.

7. Conclusion

The goal of this Master's Thesis was to provide a Proof of Concept (PoC) to demonstrate a way to implement shared cyber exercises and training between the Université Libre de Bruxelles (ULB) and Royal Military Academy (RMA). Multiple steps have been taken to provide this PoC.

First, the *State of the Art* in terms of *Cyber Range (CR)* and *Federated Cyber Range (FCR)* was addressed. The concept of CR was firstly introduced. The question of their purpose and of their usage was then addressed and answered. The requirements of such technology was identified and rigorously analyzed. The types of existing CR solutions with their advantages and disadvantages were evaluated. This phase of research allowed to gain knowledge and understanding regarding CRs and the federation aspect. Such first phase was elementary for the next phases of the work. The research process also allowed to identify the lack of clear documentation regarding concepts and projects explicitly labeled as *FCR*.

Once the knowledge and understanding gained, it was possible to address the identification of challenges encountered by FCRs. This identification phase was essential to propose a well-considered and relevant concept. To have the best approach regarding the identification of those challenges, experimentations were conducted in parallel to the theoretical research phase. Key challenges such as hypervisors, automation and connectivity were identified.

After the identification of the challenges took place the formulation of the concept proposition. The concept addressed the automation, hypervisor and connectivity aspects.

The deployments of resources were to be performed via *as-Code* tools to provide the required automation level. *Packer by Hashicorp*, *Terraform by Hashicorp* and *Ansible by Red Hat*, were chosen to fill the automation requirements. *Packer* was used to build custom VM templates, *Terraform* was the element deploying those templates, and then finally *Ansible* was used to configure the deployed resources.

The hypervisor element of the concept was to use ESXi, or nested ESXi when it was not possible to use this hypervisor directly on the hardware. Those hypervisors were to be controlled via the vCenter.

The connection between the hypervisors was delegated to Virtual Machines providing Virtual Private Network (VPN) capabilities residing on those same hypervisors. Using a central site was designated as the best option to connect sites. It was also stated that public cloud providers could help mitigate the creation of a network bottleneck at the central site.

The PoC was originally supposed to be experimented between the ULB and the Belgian RMA. Due to the COVID-19 situation at the time of implementing the PoC, such implementation was not possible. Therefore, the concept has only been implemented and tested in laboratory environment. The PoC implicated multiple host hypervisors: ESXi and KVM all running nested virtualized ESXi. The connectivity between the hypervisors was assured by pfSense VMs residing on the nested ESXi of each host.

To conclude, the PoC successfully demonstrated the ability to deploy some resources automatically on multiple nested hypervisors on multiple sites via *as-Code* tools. The limitations, performances and future work regarding this PoC and its initial concept were also addressed.

References

[1] J. Davis and S. Magrath, “A survey of cyber ranges and testbeds,” Defence science and technology organisation Edinburgh (Australia) cyber and . . ., 2013.

[2] M. Rouse. (Dec. 2018). “What is sandbox (computer security)? - definition from whatis.com,” [Online]. Available: `https://searchsecurity.techtarget.com/definition/sandbox` (visited on 04/10/2020).

[3] T. Debatty and W. Mees, “Building a cyber range for training cyberdefense situation awareness,” in *2019 International Conference on Military Communications and Information Systems (ICMCIS)*, 2019, pp. 1–6.

[4] G. M. Deckard, “Cybertropolis: Breaking the paradigm of cyber-ranges and testbeds,” in *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*, 2018.

[5] J. Vykopal, R. Ošlejšek, P. Celeda, M. Vizváry, and D. Tovarňák, “Kypo cyber range: Design and use cases,” Jan. 2017, pp. 310–321.

[6] J. Vykopal, M. Vizváry, R. Oslejsek, P. Celeda, and D. Tovarnak, “Lessons learned from complex hands-on defence exercises in a cyber range,” in *2017 IEEE Frontiers in Education Conference (FIE)*, 2017.

[7] K. Chung. (2017-2020). “Ctfd : What is capture the flag?” [Online]. Available: `https://ctfd.io/whats-a-ctf/` (visited on 10/04/2020).

[8] M. Rouse. (Jun. 2019). “What is incident response? definition from whatis.com,” [Online]. Available: `https://searchsecurity.techtarget.com/definition/incident-response` (visited on 04/11/2020).

[9] U.S. Department of the Interior. (Jun. 2015). “Penetration Testing.” Library Catalog: www.doi.gov, [Online]. Available: `https://www.doi.gov/ocio/customers/penetration-testing` (visited on 04/10/2020).

[10] M. Rouse. (Oct. 2018). “What is pen test (penetration testing)? - Definition from WhatIs.com,” [Online]. Available: `https://searchsecurity.techtarget.com/definition/penetration-testing` (visited on 10/04/2020).

[11] Z. Chen, L. Yan, Y. He, D. Bai, X. Liu, and L. Li, “Reflections on the construction of cyber security range in power information system,” in *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, IEEE, 2018, pp. 2093–2097.

[12] Secdevops-Cuse. (Mar. 2020). “Secdevops-cuse/cyberrange,” [Online]. Available: `https://github.com/secdevops-cuse/CyberRange` (visited on 04/11/2020).

[13] V. E. Urias, W. M. S. Stout, B. Van Leeuwen, and H. Lin, “Cyber range infrastructure limitations and needs of tomorrow: A position paper,” in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–5.

[14] Ansible. (Apr. 2020). “Ansible/ansible,” [Online]. Available: `https://github.com/ansible/ansible` (visited on 04/12/2020).

[15] Chef. (2008-2018). “Powerful infrastructure automation,” [Online]. Available: `https://www.chef.sh/` (visited on 04/12/2020).

[16] Hashicorp. (Apr. 2020). “Hashicorp/terraform,” [Online]. Available: `https://github.com/hashicorp/terraform` (visited on 04/12/2020).

[17] D. Kushner. (Feb. 2013). “Full page reload,” [Online]. Available: https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet (visited on 04/12/2020).

[18] C. Dictionary. (unknown). “Simulator: Meaning in the cambridge english dictionary,” [Online]. Available: https://dictionary.cambridge.org/dictionary/english/simulator (visited on 04/13/2020).

[19] ——, (unknown). “Emulator: Meaning in the cambridge english dictionary,” [Online]. Available: https://dictionary.cambridge.org/dictionary/english/emulator (visited on 04/13/2020).

[20] (unknown). “Gns3 documentation,” [Online]. Available: https://docs.gns3.com/ (visited on 04/13/2020).

[21] B. P. Tholeti. (2011). “Learn about hypervisors, system virtualization, and how it works in a cloud environment,” [Online]. Available: https://developer.ibm.com/articles/cl-hypervisorcompare/ (visited on 04/13/2020).

[22] J. Shuja, A. Gani, and S. A. Madani, “A qualitative comparison of mpsoc mobile and embedded virtualization techniques,” *arXiv preprint arXiv:1605.01168*, 2016.

[23] A. Awasthi and R. Gupta, “Multiple hypervisor based open stack cloud and vm migration,” in *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, 2016.

[24] (2020). “Edurange/edurange-server,” [Online]. Available: https://github.com/edurange/edurange-server (visited on 04/18/2020).

[25] (unknown). “What is a private cloud - definition: Microsoft azure,” [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-a-private-cloud/ (visited on 04/18/2020).

[26] (unknown). “Build the future of open infrastructure.,” [Online]. Available: https://www.openstack.org/ (visited on 04/18/2020).

[27] (unknown). “The open source cloud management platform developed for the enterprise,” [Online]. Available: https://opennebula.io/ (visited on 04/18/2020).

[28] C. Dictionary. (unknown). “Federation: Meaning in the cambridge english dictionary,” [Online]. Available: https://dictionary.cambridge.org/dictionary/english/federation (visited on 04/19/2020).

[29] (Dec. 2019). “Emulab basic concepts,” [Online]. Available: http://docs.emulab.net/basic-concepts.html (visited on 04/19/2020).

[30] (2020). “Utahhardware,” [Online]. Available: https://wiki.emulab.net/Emulab/wiki/UtahHardware (visited on 04/19/2020).

[31] (Dec. 2019). “Emulab chef tutorial,” [Online]. Available: http://docs.emulab.net/chef-tutorial.html (visited on 04/19/2020).

[32] (Aug. 2015). “Deter federation,” [Online]. Available: https://fedd.deterlab.net (visited on 04/19/2020).

[33] S. EMK. (unknown). “Project summary,” [Online]. Available: https://echonetwork.eu/project-summary/ (visited on 05/13/2020).

[34] ——, (unknown). “Press Release Kick-Off,” [Online]. Available: https://echonetwork.eu/press-releases/press-release-kick-off/ (visited on 05/13/2020).

[35] Awsdocs. (2020). “Awsdocs/aws-site-to-site-vpn-user-guide,” [Online]. Available: https://github.com/awsdocs/aws-site-to-site-vpn-user-guide/blob/master/doc_source/VPN_CloudHub.md (visited on 04/23/2020).

[36] (unknown). “World wide education roaming for research & education,” [Online]. Available: https://www.eduroam.org/ (visited on 04/25/2020).

[37] (2020). “Terraform by HashiCorp,” [Online]. Available: https://www.terraform.io/index.html (visited on 05/01/2020).

[38] HashiCorp. (Jan. 2019). “Ansible and Terraform: Better Together,” [Online]. Available: https://www.hashicorp.com/resources/ansible-terraform-better-together/ (visited on 05/01/2020).

[39] (2020). “Packer by HashiCorp,” [Online]. Available: https://www.packer.io/index.html (visited on 05/01/2020).

[40] (2020). “Providers,” [Online]. Available: https://www.terraform.io/docs/providers/index.html (visited on 04/30/2020).

[41] (2020). “All modules — Ansible Documentation,” [Online]. Available: https://docs.ansible.com/ansible/latest/modules/list_of_all_modules.html (visited on 04/30/2020).

[42] (2018). “OpenStack Docs: Choosing a hypervisor,” [Online]. Available: https://docs.openstack.org/arch-design/design-compute/design-compute-hypervisor.html (visited on 04/30/2020).

[43] (2019). “Start Here: OpenNebula Overview — OpenNebula 5.8.5 documentation,” [Online]. Available: https://docs.opennebula.io/5.8/intro_release_notes/concepts_terminology/intro.html (visited on 04/30/2020).

[44] Ahelms. (unknown). “Expose vmware hardware assisted virtualization,” [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.vm_admin.doc/GUID-2A98801C-68E8-47AF-99ED-00C63E4857F6.html (visited on 04/26/2020).

[45] (2020). “What is AWS Site-to-Site VPN? - AWS Site-to-Site VPN,” [Online]. Available: https://docs.aws.amazon.com/vpn/latest/s2svpn/VPC_VPN.html (visited on 05/02/2020).

[46] cherylmc. (2020). “Connect on-premises network to Azure virtual network: Site-to-Site VPN: Portal - Azure VPN Gateway,” [Online]. Available: https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-howto-site-to-site-resource-manager-portal (visited on 05/02/2020).

[47] (2020). “Cloud VPN overview,” [Online]. Available: https://cloud.google.com/vpn/docs/concepts/overview (visited on 05/02/2020).

[48] (Oct. 2019). “Support for running ESXi/ESX as a nested virtualization solution (2009916),” [Online]. Available: https://kb.vmware.com/s/article/2009916?lang=en_us (visited on 05/02/2020).

[49] (Apr. 2020). “Unattended Debian/Ubuntu Installation,” [Online]. Available: https://www.packer.io/guides/automatic-operating-system-installs/preseed_ubuntu/ (visited on 05/12/2020).

[50] (Apr. 2020). “User Variables - Templates,” [Online]. Available: https://www.packer.io/docs/templates/user-variables/ (visited on 05/12/2020).

[51] (May 2020). “Hashicorp/hcl,” [Online]. Available: https://github.com/hashicorp/hcl (visited on 05/05/2020).

[52] (Dec. 2019). “Do-community/ansible-playbooks,” [Online]. Available: https://github.com/do-community/ansible-playbooks (visited on 05/03/2020).

[53] (2020). “Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing, Open-Source Benchmarking,” [Online]. Available: https://www.phoronix-test-suite.com/ (visited on 05/04/2020).

[54] (unknown). “OpenBenchmarking.org - IOzone Test Profile,” [Online]. Available: https://openbenchmarking.org/test/pts/iozone (visited on 05/05/2020).

[55] (unknown). “OpenBenchmarking.org - 7-Zip Compression Test Profile,” [Online]. Available: https://openbenchmarking.org/test/pts/compress-7zip (visited on 05/05/2020).

[56] (Mar. 2020). “Instructions per second,” [Online]. Available: https://en.wikipedia.org/w/index.php?title=Instructions_per_second&oldid=946747290 (visited on 05/06/2020).

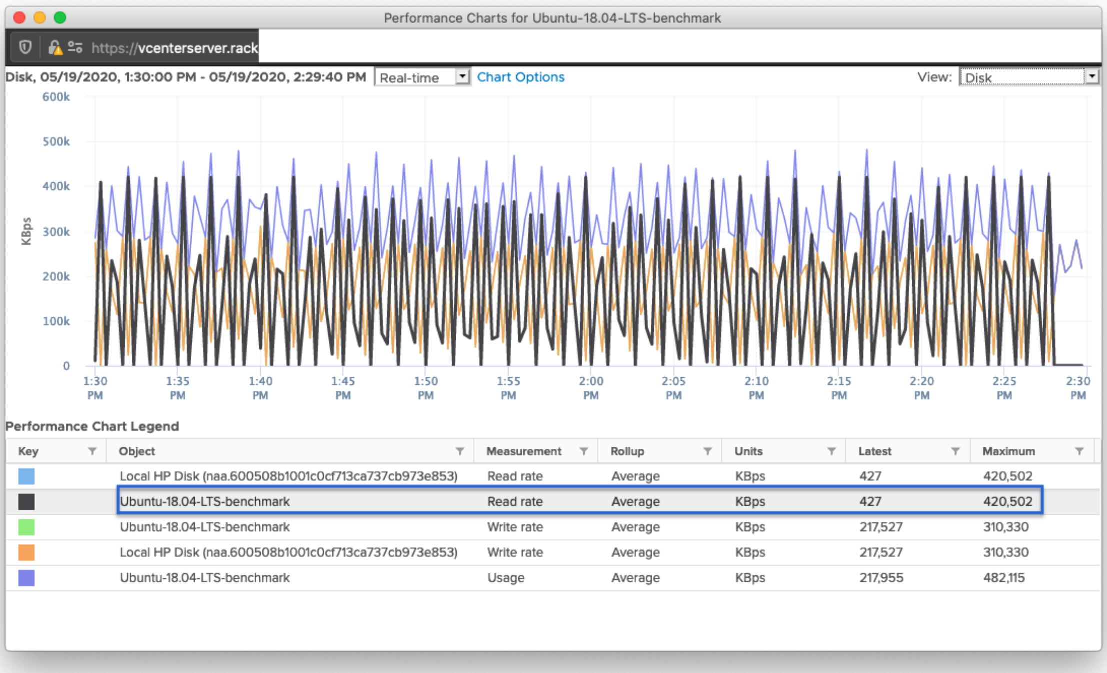
[57] (unknown). “OpenBenchmarking.org - RAMspeed SMP Test Profile,” [Online]. Available: https://openbenchmarking.org/test/pts/mbw (visited on 05/05/2020).

[58] (Apr. 2020). “Guest os customization support matrix,” [Online]. Available: https://partnerweb.vmware.com/programs/guestOS/guest-os-customization-matrix.pdf (visited on 05/02/2020).

[59] (2020). “Changelog – Oracle VM VirtualBox,” [Online]. Available: https://www.virtualbox.org/wiki/Changelog (visited on 05/05/2020).

A. ESXi monitoring

A.1 Standard virtualization



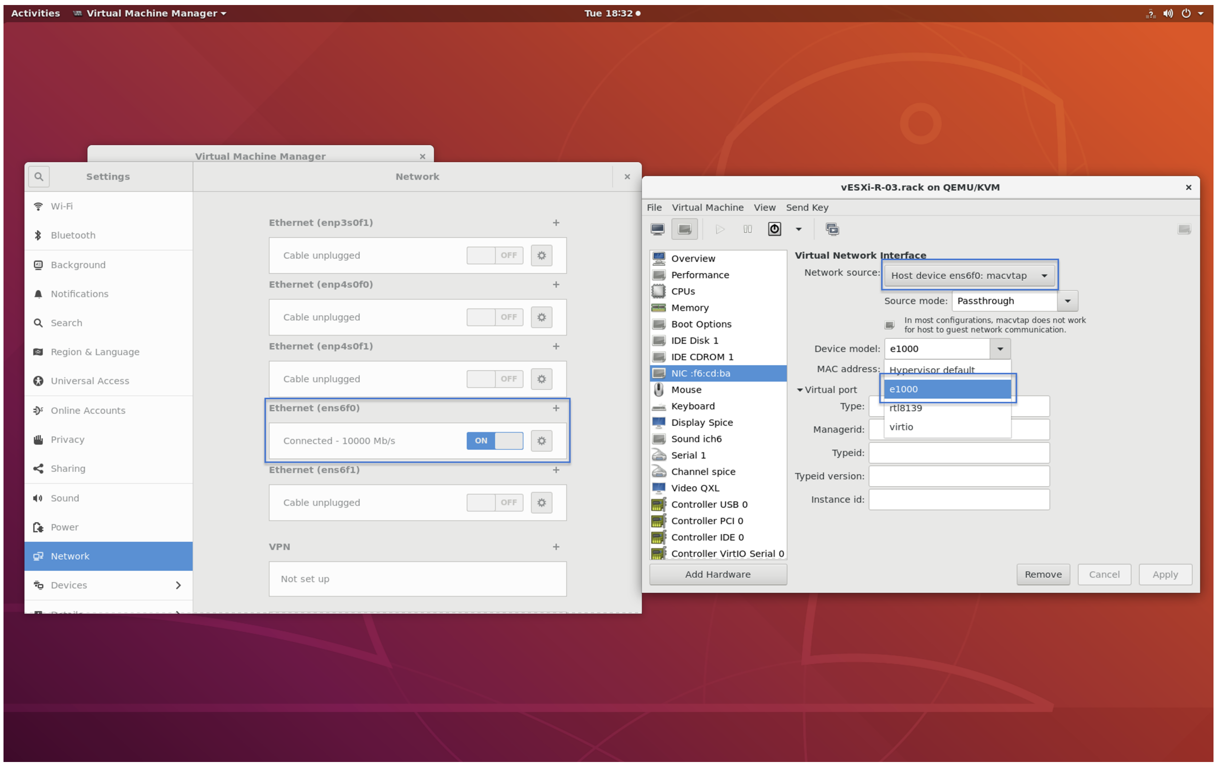
Performance Chart Legend

Key	Object	Measurement	Rollup	Units	Latest	Maximum
🟦	Local HP Disk (naa.600508b1001c0cf713ca737cb973e853)	Read rate	Average	KBps	427	420,502
⬛	Ubuntu-18.04-LTS-benchmark	Read rate	Average	KBps	427	420,502
🟩	Ubuntu-18.04-LTS-benchmark	Write rate	Average	KBps	217,527	310,330
🟧	Local HP Disk (naa.600508b1001c0cf713ca737cb973e853)	Write rate	Average	KBps	217,527	310,330
🟪	Ubuntu-18.04-LTS-benchmark	Usage	Average	KBps	217,955	482,115

Figure A.1: ESXi built-in monitoring - disk read benchmark.

B. KVM NIC limitation

B.1 KVM side



Ethernet (enp3s0f1)
Cable unplugged OFF

Ethernet (enp4s0f0)
Cable unplugged OFF

Ethernet (enp4s0f1)
Cable unplugged OFF

Ethernet (ens6f0)
Connected - 10000 Mb/s ON

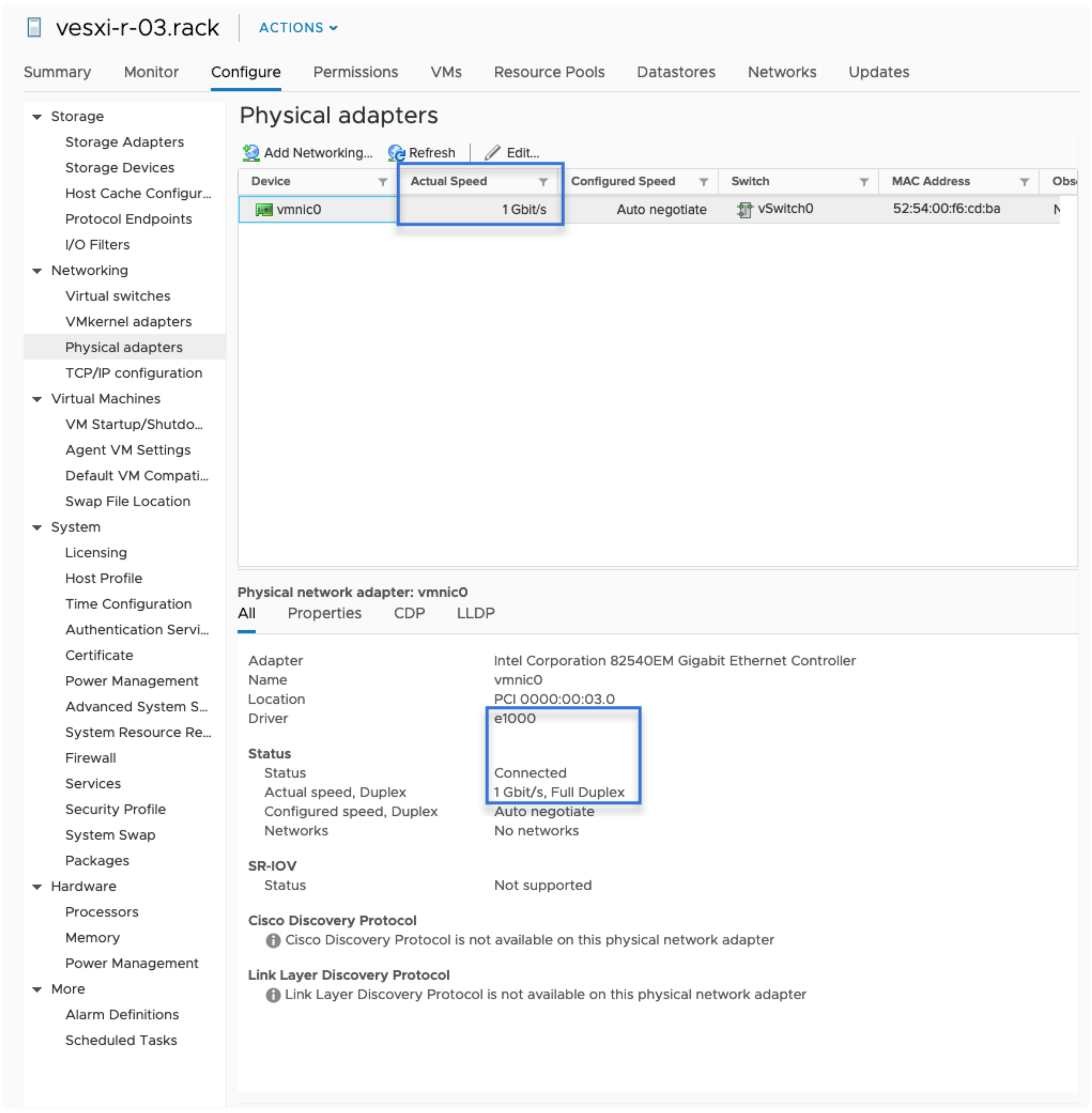
Ethernet (ens6f1)
Cable unplugged OFF

VPN
Not set up

Virtual Network Interface
Network source: Host device ens6f0: macvtap
Source mode: Passthrough
In most configurations, macvtap does not work for host to guest network communication.
Device model: e1000
MAC address: Hypervisor default
e1000
rtl8139
virtio
Virtual port
Type:
Managerid:
Typeid:
Typeid version:
Instance id:

Figure B.1: KVM limitation system.

B.2 vCenter side



Device	Actual Speed	Configured Speed	Switch	MAC Address	Obs
vmnic0	1 Gbit/s	Auto negotiate	vSwitch0	52:54:00:f6:cd:ba	N

Physical network adapter: vmnic0

All Properties CDP LLDP

Adapter	Intel Corporation 82540EM Gigabit Ethernet Controller
Name	vmnic0
Location	PCI 0000:00:03.0
Driver	e1000
Status	
Status	Connected
Actual speed, Duplex	1 Gbit/s, Full Duplex
Configured speed, Duplex	Auto negotiate
Networks	No networks
SR-IOV	
Status	Not supported

Cisco Discovery Protocol
Cisco Discovery Protocol is not available on this physical network adapter

Link Layer Discovery Protocol
Link Layer Discovery Protocol is not available on this physical network adapter

Figure B.2: RAM speed of KVM system.