

Tallinn University of Technology
School of Information Technology
Department of Software Science

ITC70LT

Kristjan Kaunis 204803IVCM

**HYPERVERSOR AGNOSTIC SCENARIO
DEFINITION LANGUAGE FOR CYBER
RANGES**

Master's thesis

Supervisors: Dr.comp.sc. Bernhards Blumbergs

Toomas Lepik, MSc

Tallinn 2022

Tallinna Tehnikaülikool
Infotehnoloogia teaduskond
Tarkvarateaduste instituut

ITC70LT

Kristjan Kaunis 204803IVCM

**HÜPERVIISORIST SÕLTUMATU
STSENAARIUMI KIRJELDUSKEEL
KÜBERHARJUTUSVÄLJADELE**

Magistritöö

Juhendajad: Dr.comp.sc. Bernhards Blumbergs

Toomas Lepik, MSc

Tallinn 2022

Declaration of originality

I declare that this thesis is the result of my own research except as cited in the references. This thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Author: Kristjan Kaunis

Date: 16 May 2022

Abstract

Creating complex cyber defense exercises take a lot of research and development effort. As such, it would be beneficial, if the developed exercises or some of its' parts could be easily shared between different cyber ranges. This can be a complex matter as cyber ranges that host those exercises often vary in terms of hardware, automation software and virtualization platform used. This research presents a novel framework, which can be used to define cyber defense exercises in a hypervisor agnostic manner. Cyber defense exercises developed in accordance with this framework would be deployable to any hypervisor platform, which has been integrated into the framework, thus greatly enhancing exercise portability between different cyber ranges.

First part of this thesis focuses on developing a hypervisor agnostic scenario definition language concept and structure. In that part, the author also describes, how to define exercise scenarios with the given framework. The second part of this thesis describes, how different hypervisors can be integrated into the framework and how exercises, which are developed following this framework, can be deployed into different hypervisors. In the final part of the thesis, the author evaluates the framework by implementing a proof of concept and deploying a small exercise environment onto KVM and VMware hypervisors. The successful deployment of the exercise proved, that by using the given framework, it is possible to develop cyber defense exercises in a hypervisor agnostic manner.

This thesis is written in English and contains 67 pages of text, eight chapters and 18 figures.

Annotatsioon

Küberkaitseõppuste tehniline arendamine on aja- ja ressursimahukas töö. Sellest tulenevalt oleks kasulik, kui kord juba välja arendatud õppuseid või nende osiseid saaks erinevate küberharjutusväljade vahel jagada. See võib aga problemaatiliseks osutuda, kuna erinevad küberharjutusväljad kasutavad tihti eri riistvara, virtualiseerimis- ja automatiseerimistarkvara. Kui õppuse jaoks loodav sisu - virtuaalmasinad ning nende juututus- ja konfiguratsioonikoodid on spetsiifiliselt mõnda kindlat hüperviisorit ja automatiseerimistarkvara silmas pidades arendatud, on nende kasutuselevõtt teises küberharjutusväljas raskendatud. Vastav küberharjutusväli peaks uue õppuse osiste kasutuselevõtuks oma enda automatiseerimistarkvara muutma ja/või hakkama esialgset konfiguratsioonikoodi muutma.

Selle probleemi vältimiseks on antud töö autor loonud uue raamistiku küberkaitseõppuste arendamiseks ja juurutamiseks kus küberkaitseõppuseid kirjeldatakse hüperviisorist sõltumatul kujul.

Lõputöö esimene pool keskendub hüperviisorist sõltumatu stsenaariumi kirjelduskeele kontseptsioonide ning struktuuri defineerimisele ja arendamisele. Teine pool tööst kirjeldab, kuidas antud raamistiku alusel kirjeldatud õppuseid erinevatesse hüperviisoritesse juurutada ning kuidas uusi hüperviisoreid antud raamistikku kasutusele võtta. Käesolevat kontseptsiooni tõestust hinnati VMware ja KVMi hüperviisoreid kasutades, juurutades mõlemasse väikese õppuse keskkonna, mis oli defineeritud antud raamistiku alusel hüperviisorist sõltumatus stsenaariumi kirjelduskeeles. Õppuse edukas juurutus mõlemasse hüperviisorisse tõestades, et küberkaitseõppuseid saab antud raamistiku alusel hüperviisorist sõltumatul kujul kirjeldada.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 67 leheküljel, kaheksat peatükki ja 18 joonist.

List of abbreviations and terms

ADLES	Automated Deployment of Lab Environment System
AIT	Austrian Institute of Technology
API	Application Programming Interface
CDX	Cyber Defense Exercise
CPU	Central Processing Unit
CRACK	Cyber Range Automated Construction Kit
CRATE	Cyber Range and Training Environment
CTF	Capture the Flag
CyRIS	Cyber Range Instantiation System
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DRS	Distributed Resource Scheduler
DSL	Domain Specific Language
DSRM	Design Science Research Methodology
ECSSO	European Cyber Security Organization
FQDN	Fully Qualified Domain Name
HCI	Hyperconverged Infrastructure
HCL	HashiCorp Language
ICS	Industrial Control Systems
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation

KVM	Kernel Virtual Machine
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MIT	Massachusetts Institute of Technology
MS	Microsoft
NATO	North Atlantic Treaty Organization
NIST	National Institute of Standards and Technology
OASIS	the Organization for the Advancement of Structured Information Standards
OS	Operating System
RAM	Random Access Memory
RDP	Remote Desktop Protocol
SCADA	Supervisory Control and Data Acquisition
SDL	Scenario Definition Language
SDN	Software Defined Networking
SID	Security Identifier
SSH	Secure Shell protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
WinRM	Windows Remote Management
WSL	Windows Subsystem for Linux
XML	Extensible Markup Language
YAML	<i>YAML ain't Markup Language</i>

Table of contents

1	Introduction	12
1.1	Motivation and problem statement	12
1.2	Research questions.....	13
1.3	Scope of the thesis	13
1.4	Ethics	14
1.5	Research methods	15
2	Current situation and related research	17
2.1	Literature review.....	17
2.2	Novelty and personal contributions	23
3	Scenario Definition Language concepts	24
3.1	Hypervisor agnostic approach.....	24
3.2	SDL language selection	24
4	Scenario Definition Language structure	27
4.1	SDL hypervisor communication method selection.....	27
4.2	Scenario definition.....	28
4.2.1	SDL environment definition.....	31
4.2.2	SDL exercise definition.....	34
4.3	Roles	36
4.3.1	Core roles	37
4.3.2	Reusable configuration.....	38
4.3.3	Virtual machine specifications	38
4.4	Virtual machine development lifecycle	41
4.5	Introducing new operating systems	42
5	Hypervisor communication	43
5.1	Environment requirements.....	43
5.1.1	Networking.....	43

5.1.2 Storage.....	44
5.1.3 Compute resources	44
5.2 Templates.....	46
5.2.1 VMware virtual machine templates	47
5.2.2 OpenNebula virtual machine templates	49
5.3 Targeting hypervisors	50
5.3.1 VMware vSphere.....	50
5.3.2 OpenNebula.....	51
5.4 Integrating new hypervisors.....	52
5.5 Deployment.....	53
6 Proof of concept implementation	56
6.1 VMware vSphere	59
6.2 OpenNebula	61
7 Discussion.....	64
8 Conclusion.....	66
References	68
Appendix 1: Ansible playbook for vSphere deployment	74
Appendix 2: SDL_engine.py	75
Appendix 3: vSphere deployment example.....	77
Appendix 4: OpenNebula deployment example.....	84
Appendix 5: vSphere dual deployment example.....	91

List of figures

Figure 1: DSRM – Design and Development Centered approach [14].....	15
Figure 2: Taxonomy of cyber ranges [20].....	19
Figure 3: YAML vs JSON syntax [45].....	26
Figure 4 : <i>SDL_environment.yml</i> , populated with example values	33
Figure 5: An example VM definition in <i>SDL_exercise.yml</i>	35
Figure 6: Ansible Roles	36
Figure 7: Ansible <i>get_url</i> module example	39
Figure 8: Ansible <i>mysql_db</i> module example	40
Figure 9: Ansible <i>apt</i> module example	40
Figure 10: Ansible <i>command</i> example	40
Figure 11: VM development lifecycle.....	41
Figure 12: OS templates in <i>SDL_environment.yml</i>	42
Figure 13: Compute resources in vSphere appliance	45
Figure 14: Compute resources in OpenNebula appliance	46
Figure 15: Hypervisor role in <i>vsphere.yml</i> playbook	53
Figure 16: Network map.....	57
Figure 17: Deployed exercise in VMware vSphere.....	60
Figure 18: Deployed exercise in OpenNebula.....	63

1 Introduction

1.1 Motivation and problem statement

Cyber ranges are becoming increasingly popular in digitalized societies [1]. Both public and private sectors are utilizing the service to train and advance their specialists in detecting and reacting to a wide array of different situations and threats through different courses and complex cyber defense exercises (CDX). Exercise scenarios have to be tailored not only for technical experts to defend their systems, but also other involved experts, such as forensic investigators, communication specialists, managers and decision makers. All of this means that a modern CDX, once built and deployed, is a massive combination of information, that could be beneficial to share and exchange.

If a cyber range in Country A has virtualized, for example, a Microsoft Active Directory Domain Environment and a cyber range in Country B has virtualized a power plant Supervisory Control and Data Acquisition (SCADA) system, then they could exchange their deployment and configuration code. That would save Country B development time needed to figure out, how to install and configure directory service for central login management in the form of Active Directory, leaving them just the work needed to integrate it with their SCADA system. From Country As perspective, they could integrate an already tested virtual SCADA system into their own exercise environment, thus saving the time needed to research and develop the respected system.

The problem in this example often boils down to cyber range hardware differences. Key assumption of this thesis is that cyber ranges differ from each other in terms of hardware selection, virtualization layer and automation software in use. For example, the SCADA system could have been deployed and configured on Kernel-based Virtual Machine (KVM) [2] and Microsoft Active Directory Domain environment on VMware ESXi [3].

Another aspect of cyber ranges is that exercises can be and often are defined using a different approach. There is a variety of automation and provisioning software available, like Ansible [4], Terraform [5], Vagrant [6], and Chef [7] to name a few. The problem is that they all are designed for a different task. For example, Ansible can be categorized as a tool for configuration management that follows a procedural approach, while Terraform can be viewed as a provisioning tool that follows a declarative approach [8]. Country A may have built up their deployment scripts using mainly Ansible, while Country B could be using Vagrant and Terraform. If Country A's exercise development and deployment code is now imported to Country B, the latter has to modify either their own deployment methods or start changing the code. This is time consuming and error prone.

To overcome aforementioned problems, the author proposes a novel framework which would enable easier exercise content portability between different Cyber Ranges. In this framework, CDXs would be defined in a generic hypervisor agnostic language, meaning that exercise development and configuration code would not contain any hypervisor specific information.

1.2 Research questions

The following thesis aims to answer to the following research questions:

1. What are the concepts for designing a Scenario Definition Language which would enable the provisioning and configuration of the same cyber defense exercise on different hypervisors?
2. What are the necessary components and syntax definitions of the hypervisor agnostic Scenario Definition Language?
3. How different hypervisors can be implemented into the Scenario Definition Language?

1.3 Scope of the thesis

In this thesis a Scenario Definition Language (SDL) proof of concept will be developed. In order to validate the deployments against different hypervisors, a Microsoft Active Directory

based exercise environment will be developed. The exercise environment will be described in more detailed in section 6.

This SDL proof of concept will be evaluated against VMware vSphere [9] (using VMware ESXi type 1 hypervisor) and OpenNebula [10] (using KVM type 1 hypervisor). A type 1 hypervisor, also called as bare-metal hypervisor, “[...] is virtualization software that has been installed directly onto the computing hardware” [11]. The goal is to prove that the exercise development and configuration information can be defined without any hypervisor specific specifications. The SDL has to contain the virtual machine (VM) configuration information, manage the communication with the hypervisors in order to deploy the VMs and manage the communication with deployed VMs for configuration.

As mentioned above, this thesis will be limited to two on-site datacenter solutions, meaning that public cloud service providers, such as, Amazon Web Services [12] and Microsoft Azure [13] will remain out of scope. The reason for this is that the additional value of validating the SDL against more than two hypervisors would be minimal with the current setup since it would only display how different SDL hypervisor connections would be built. As every new hypervisor integration takes its’ time, it would shift the main research focus from the actual SDL research and development to hypervisor connector building. Research questions may be answered with two aforementioned on-site datacenter solutions. The developed SDL will be evaluated through a proof of concept which targets two previously mentioned hypervisors and guidance will be provided, how additional hypervisors can be integrated.

1.4 Ethics

The given thesis focuses on creating an SDL proof of concept, which could be used later by other interested parties to develop it further. This given thesis will not contain any sensitive personal data nor any sensitive information about the production environments. All the detailed hostnames and logon information will be generalized.

1.5 Research methods

The given thesis will follow the Design Science Research Methodology (DSRM) framework with a Design- and Development-Centered Approach, which was chosen since it provides a structured approach on producing and evaluating a novel artifact [14].

The DSRM consists of six steps:

1. Problem identification and motivation (section 1).
2. Define the objectives of a solution (sections 1 and 3).
3. Design and development (sections 4 and 5).
4. Demonstration (section 6).
5. Evaluation (sections 6 and 7).
6. Communication (section 8).

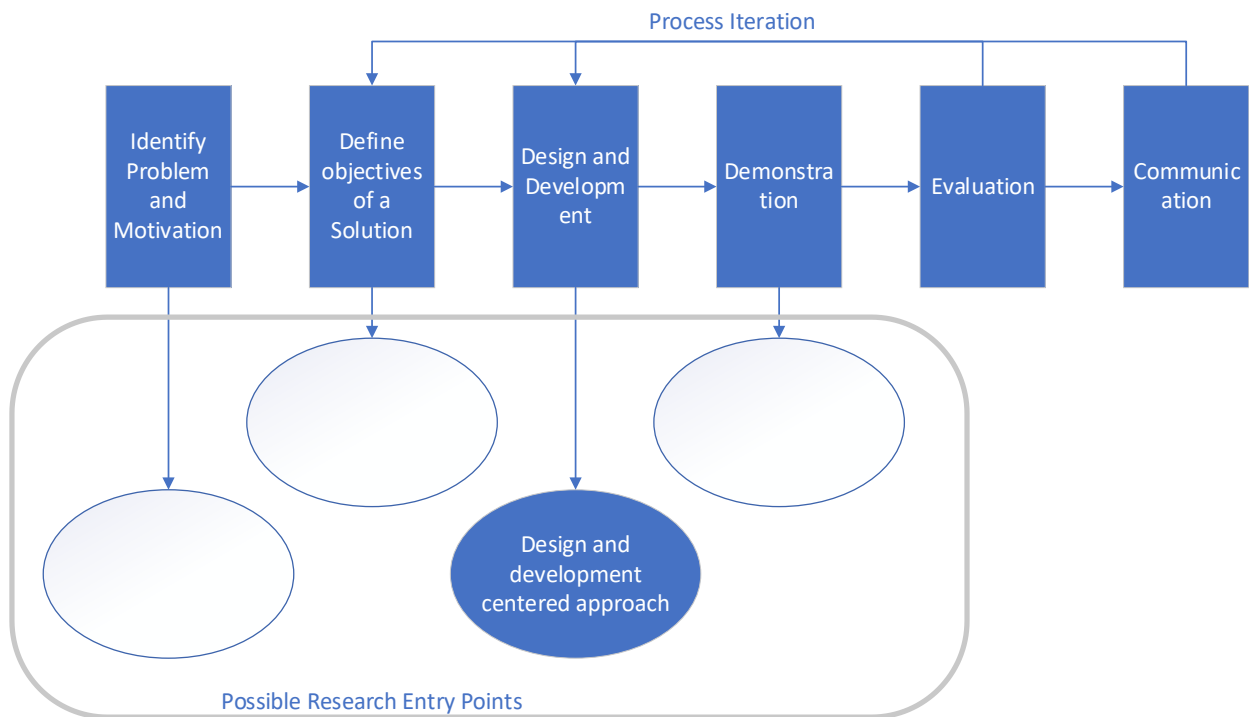


Figure 1: DSRM – Design and Development Centered approach [14]

The produced artifact's usefulness will be evaluated by creating a CDX scenario with parameters described in section 6, and validating that deployment produces equal outcomes in both VMware and OpenNebula environments without modifying the CDX content. After evaluation, key takeaways of the proof of concept will be addressed for future development.

2 Current situation and related research

In this section, the author will perform a literature review on existing scenario definition languages and cyber ranges. After that, the author will highlight the novelty of this thesis by pointing out established research gaps and describing the personal contributions.

2.1 Literature review

Literature review for this thesis was conducted in two phases. The first phase focused on gathering information about layouts of different existing cyber ranges. In that phase the author tried to gain an understanding, how current state-of-the-art cyber ranges are built and how they differ from each other in terms of virtualization layer and hardware/software choices. In the second phase, the author performed a literature review on cyber range automation, scenario definition and development techniques in use.

The main digital library for the literature review was Scopus. [15] Searches included keyword combinations of:

Cyber + range, cyber + range + scenario, cyber + range + language, SDL + scenario + language, windows + domain + scenario, design + validation + scenario + cyber, windows + domain + automation, hypervisor + cyber + range, hypervisor + agnostic, hypervisor + language.

Those keywords were chosen to find possible results regarding scenario definition languages and cyber ranges All of the search results were limited to last five years.

For the second phase, aside from the results found during the previously mentioned keyword searches, the author used backwards snowballing research method, with the initial seed of the studies being “Building next generation Cyber Ranges with CRACK”. [16]

To begin with, the author identified, that the term cyber range is loosely defined and may have a very broad meaning. For example, regarding National Institute of Standards and Technology (NIST), cyber ranges are defined as “[...] interactive, simulated representations of an organizations local network, system, tools and applications that are connected to a simulated internet level environment” [17]. European Cyber Security Organization (ECSO) defines cyber ranges as a platform, which “[...] can be intended to be a group of technologies that are used to create and use a simulation environment” [18]. The latter definition is broader than of the NIST, pointing out that cyber ranges are not just for simulating office environments, but rather function as testbeds for any kind of virtualized technologies. Diateam describes cyber range as a “[...] virtual environment that enables organizations to simulate cyber combat training, system/network development, testing and benchmarking.” [19] As can be concluded, ECSO definition focuses more on the hardware side of the Cyber Ranges, while Diateam focuses more on the provided services. Since many cyber ranges are built with a different goal in mind, it is difficult to come up with a definition, that would cover the use cases of every existing cyber range. Because the scope of this thesis focuses on the virtualization layer and automation software selections, the author will follow the ECSO definition of the cyber range.

An academic research paper [20] about cyber ranges was conducted by the Norwegian University of Science and Technology. Based on their systematic literature review, they have developed a taxonomy to classify cyber ranges.

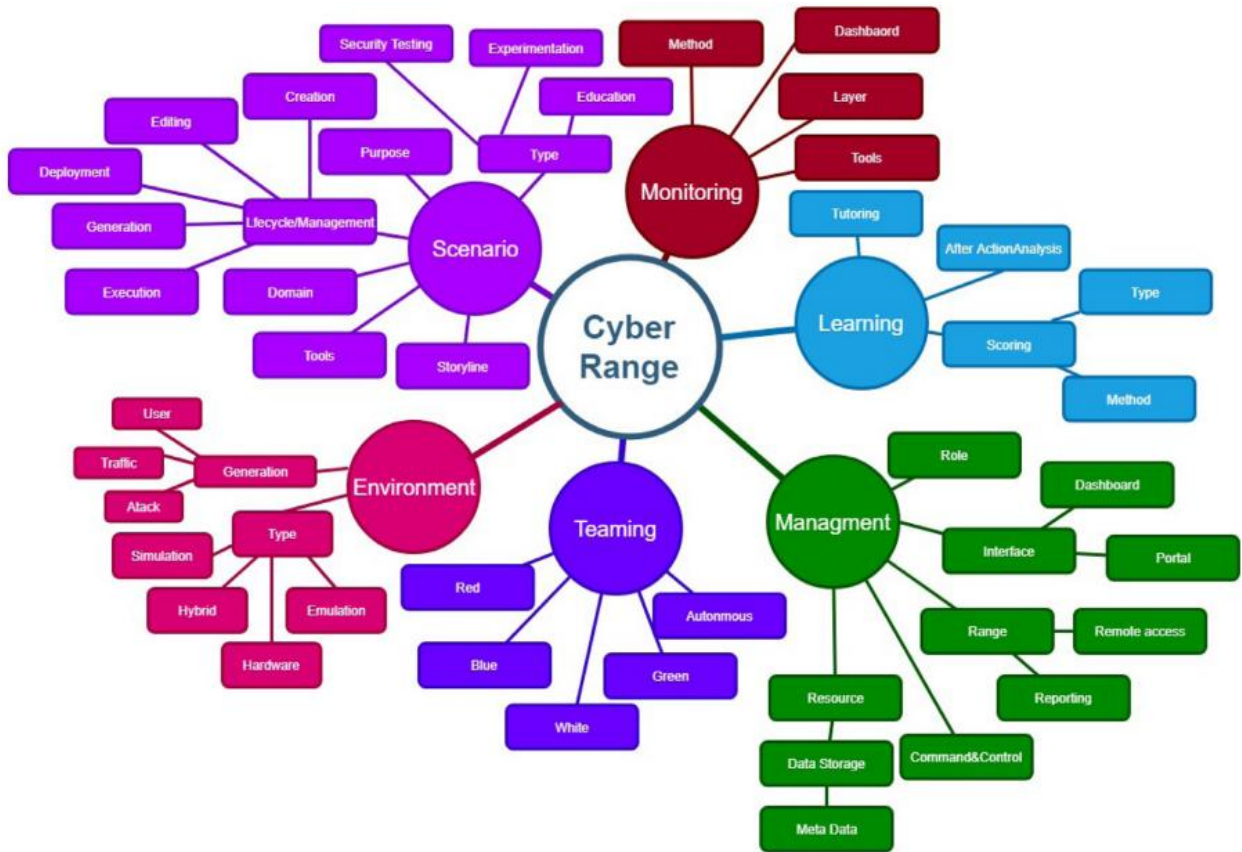


Figure 2: Taxonomy of cyber ranges [20]

Figure 2: Taxonomy of cyber ranges shows the amount of different information a cyber range can contain. Each section of the presented taxonomy was analyzed and described by the authors, thus giving a comprehensive overview. That taxonomy illustrates what concepts in a cyber range can be considered part of a scenario. This is a comprehensive, high abstraction overview and by no means do every CDX include every aspect from the given Taxonomy.

A research paper from the Swedish Defense Research Agency presented a case study of Cyber Range and Training Environment (CRATE) [21] from the Cyber Range automation perspective. What sets this paper apart from the rest is the use of a custom operating system (OS), called CrateOS, which is used by the virtualization servers. CRATE makes use of a list of custom-built automation features, which include but are not limited to Range Provisioning, System and Service Configuration and Exercise Management to name a few.

There was also a notable amount of gray literature and white papers about cyber ranges. A white paper on KYPO cyber range [22] gives an overview of the Czech Republic largest

academic cyber range, built on OpenNebula. Also, a gray paper from University of Tartu explains how to federate the KYPO cyber range with the Estonian National cyber range, which is powered by VMware technologies [23]. Both of these papers give insight how different cyber ranges work and what kind of solutions are possible to implement.

Another white paper on Austrian Institute of Technology (AIT) cyber range [24] gives an overview of a cyber range, “[...] which was designed based on several principles, such as scalability, flexibility and the utilization of Open Source technologies” and covers different aspects of its architecture and provisioning techniques.

Lastly, a different approach on cyber ranges was presented by the University of Coimbra via a SCADA Industrial Control Systems (ICS) cyber range [25]. Besides giving an in-depth overview of their cyber range physical layout, the paper also focuses on different security training aspects from the SCADA ICS perspective.

The aforementioned papers focus mostly on the physical and technical side of operating different cyber ranges.

The second phase consists of gathering information on automation, scenario definition and development. Due to the previously mentioned vague definition, the literature about cyber ranges varies a lot in approach, complexness and outcomes.

To begin with, there are three studies [16], [26], [27] which try to achieve similar objectives as this thesis. They all have their limitations and drawbacks though, because of their scope and approach to the problem.

The first paper presented a Cyber Range Automated Construction Kit (CRACK) [16] which focuses on the design, model verification, generation, and automated testing of cyber scenarios. In their work, one of the questions addressed by the authors was: “[c]an designers reuse (parts of) previously developed scenarios without compromising the overall quality of the training activity?” [16].

The drawback of that paper was that CRACK framework is based on Topology and Orchestration Specification for Cloud Applications (TOSCA), or – to be more exact, it was an extension of TOSCA. “TOSCA is an Organization for the Advancement of Structured Information Standards (OASIS) open standard that defines the interoperable description of

services and applications hosted on the cloud and elsewhere; including their components, relationships, dependencies, requirements, and capabilities [...]”. [28]

This means that the underlying infrastructure must be TOSCA compatible, thus setting a limitation the author wishes to avoid in this thesis. There are many different TOSCA orchestrators and they all have different features. For example, Cloudify Private Cloud Orchestration seems to only cover OpenStack and VMware Suite [29]. The idea with a hypervisor agnostic SDL is that all the necessary toolset required comes from the SDL development and no specific software is required from the target cyber range, guaranteeing that all the defined specifications in the SDL are properly applied.

A strong side of the CRACK paper is that it “supports the *(i)* design, *(ii)* automated verification and *(iii)* deployment and *(iv)* automated testing of complex Cyber Range Scenarios.” [16] Although CRACK is based on TOSCA, their SDL covers desired steps of design, verification and deployment. Some of those approaches can be tested in this thesis, when the TOSCA attributes are modified.

The second paper presented an Automated Deployment of Lab Environment System (ADLES) [26]. In this paper the authors aim to create a “[...] formal specification language that enables the complete and self-contained formal description of virtual educational environments.” [26] In terms of SDL, the paper gives valid examples, how exercise metadata and environment information can be defined. The paper also focused on creating an open source tool-set “[...] that is capable of semi-automatically creating and deploying hands-on exercise environments described by the ADLES specification.” [26] The downside of that paper was that hypervisor specific information was still written into the ADLES specifications. Also, as the whole tool is designed for VMware vSphere, it can take a lot of time and effort to get all the functions working on different hypervisors. In the ADLES paper, adding support for additional hypervisors is mentioned as one of the future works. That is something the author of this thesis tries to address from the beginning – by designing a tool to be hypervisor agnostic in its core and clearly stating, how to connect the tool to different hypervisors. However, the paper provides very good examples how to define and design the user-facing part of the SDL.

The third paper “[...] developed a multi-layer system (toolset) to support the planning and execution of cybersecurity exercises” [27]. The developed toolset consists of three different parts, namely a cyber security strategy game, domain specific language (DSL) and infrastructure orchestration module. In first part, an exercise scenario and topology are modelled by exercise designers, using a drag and drop interface [27]. DSL is then used to generate a YAML file based on the input received from the previous part [27]. Finally, the orchestration module performs syntax validation and upon receiving no errors, translates YAML into Heat templates [30] and Puppet [31] stack, which are then used to deploy the exercise into an OpenStack cloud environment [27].

In theory, that’s a very similar functionality to what this thesis is aiming to achieve. The main downside of the mentioned paper is that it is not hypervisor agnostic, meaning that it is only developed to work in conjunction with OpenStack. This does not allow defined exercise portability between other cyber ranges which do not use OpenStack as the chosen method for hypervisor communication via Heat templates is not implementable on a range of hypervisors. In this thesis the author prioritizes exercise portability between cyber ranges with different hypervisors.

Besides those previously mentioned papers, there was a list of additional relevant literature.

Cyber Range Instantiation System (CyRIS) [32], [33] introduces an open source tool for Cybersecurity Education and Training Support. Although CyRIS is not hypervisor agnostic, it does provide yet another overview of a possible approach on automating cyber range deployments.

A more model-based approach was proposed in a paper which focuses on Security Assurance Modelling. [34] While not being overly technical, “[t]his comprehensive approach allows us to identify and describe the assets of the system, their relations and their corresponding threats; the sequence of events that leads to the manifestation of these threats, alongside the responsible threat actor/s [...]” [34], which can be useful when designing the SDL from security or inject and vulnerability perspective.

Other identified work in the area of cyber ranges and Scenario Development/Definition Languages focused on Capture-the-Flag (CTF) and Hack-the-Box exercise developments

and deployments. Three papers [35], [36], [37] focused on cyber ranges from a CTF perspective, and while not providing any direct value to SDL development, they give an overview about challenges that lie in CTF exercise design. For example, one of those papers [37] focused on reducing the resources needed and simplifying the CTF infrastructure setup for the organizers through the use of application containers instead of virtual machines. Another paper [35] provided a list of lessons learned from hosting a 317 team, 24-hour Attack and Defense style CTF in Amazon Web Services.

These papers give an overview of the current status of different cyber range related problems, obstacles and solutions. As cyber security training in different forms, such as CTFs and CDXs gain popularity, new development and automation approaches are being integrated into cyber ranges. Although there is a variety of literature on different cyber ranges and CDXs with their respective use-cases, the author did not find any literature, where exercise portability between different cyber ranges was prioritized.

2.2 Novelty and personal contributions

At the time of the review and based on the identified criteria, the author did not find any studies about development of a hypervisor agnostic scenario definition language. This proof of concept builds towards a new approach, where exercise content can be defined without any hypervisor specific information, thus bringing exercise portability to a new level. This would be a big leap forward in terms of sharing technical information in a field where a lot of content is constantly being created.

The personal contributions delivered through this thesis are the following:

- thorough existing SDL implementation analysis within the performed literature review (section 2);
- SDL prototyping and hypervisor agnostic design considerations (sections 3 and 4);
- SDL implementation requirements on different hypervisors: (section 5);
- evaluation of the SDL on KVM and VMware hypervisors (section 6).

3 Scenario Definition Language concepts

In this chapter, the author first describes the requirements for the hypervisor agnostic SDL. The author will then perform a review on data serialization languages in order to choose a data format for the SDL. Different data serialization languages will be compared to see, which one meets the defined criteria.

3.1 Hypervisor agnostic approach

For the SDL to be hypervisor agnostic, hypervisor specific information must be avoided in the exercise environment and VM definitions. The SDL must provide a method for adding dedicated hypervisor communication methods to deploy the VMs into different hypervisor environments, while the rest of the exercise configuration would be defined with the assumption that the underlying VMs are already deployed and running on a hypervisor platform. This approach would enable to define exercise content on one hypervisor which would then be applicable on other additional hypervisors, once the communication methods are established. During scenario definition, developed VMs must contain information about the underlying OS requirements, which will be interpreted by the SDL during exercise deployment.

3.2 SDL language selection

The criteria for the SDL language selection are to be human and machine readable, logical and succinct. Based on the literature review, the most common languages for existing scenario definition languages have been JavaScript Object Notation (JSON) [38] and YAML [39]. Since the goal is to use SDL to define values for specific attributes, which can be then parsed by the SDL engine (described in more detail in section 4.2), the list of potential languages is not very long, when bearing in mind the requirement of human readability.

Besides the previously mentioned two languages, there are also Extensible Markup Language (XML) [40], MessagePack [41] and protobuf [42].

MessagePack will not be considered a viable option since it can be classified as a lightweight version of JSON, which makes it fast, but in expense of human readability. The same is the case with Protobuf, an open source serialization method created by Google. These two data serializers specialize in bringing down the message size and response times, which are good qualities, but regarding current thesis, cutting down microseconds per actions is not as relevant as creating a user-friendly, easy-to-understand framework for the SDL.

XML is a markup language, compared to JSON and YAML, which are data serialization languages. “A markup language is a computer language that uses tags to define elements within a document.” [43] Data serialization, on the other hand is “[...] the process of converting data objects present in complex data structures into a byte stream for storage, transfer, and distribution purposes on physical devices.” [44]

XML, JSON and YAML are all:

- self-describing (human readable);
- hierarchical, meaning they can contain values within values;
- parsed and used by many programming languages.

But on the other side, when for example comparing XML to JSON, the latter syntax is shorter, due to not using end tags, thus being quicker to write and easier on the eye. Also, JSON can use arrays, which is not the case for XML. Another thing to bear in mind is that XML has to be parsed with an XML parser, while JSON can be parsed by a standard JavaScript function. When comparing XML to YAML, similar differences can be observed. YAML syntax is much less verbose, thus being easier to read and edit. As can be concluded from those two comparisons, XMLs main disadvantage is its syntax, which is too verbose compared to its competitors.

In order to choose between JSON and YAML, both languages were studied to assess their suitability. They are both data serialization languages that use key – value pairs. YAML is a superset of JSON, meaning that every JSON file is also a valid YAML file. As displayed in figure 3: YAML vs JSON syntax, the syntax is slightly different. In terms of human

readability, YAML has an advantage, not just because of the fact that its syntax requires fewer special characters, but also because the language allows commenting, a feature that is missing in JSON. The latter is a desired feature for open-source frameworks, as it provides developers a convenient way to explain and reason the code.

YAML	JSON
<pre>simple-property: a simple value object-property: a-property: a value another-property: another value array-property: - item-1-property-1: one item-1-property-2: 2 - item-2-property-1: three item-2-property-2: 4 # no comment in JSON</pre>	<pre>{ "simple-property": "a simple value", "object-property": { "a-property": "a value", "another-property": "another value" }, "array-of-objects": [{ "item-1-property-1": "one", "item-1-property-2": 2 }, { "item-2-property-1": "three", "item-2-property-2": 4 }] }</pre>

Figure 3: YAML vs JSON syntax [45]

It is difficult to assess the community size of both, since neither of them are programming – but rather data serialization languages. But in general, JSON seems to have a bigger community support of the two [45] [46].

YAML has been criticized for being hard to edit, when file sizes grow [47]. The author of the said article points out that as files and functions grow in depth, the indentation, which is two spaces, instead of one tab, can grow difficult to follow. While this is subjective to how and what kind of code is written, the article later points out that the YAML statement regarding easy human readability is true if “[...] one stick[s] to a small subset” [47]. As the goal of the SDL is to keep the user input at minimum, this criticism, in the authors opinion, does not stand.

The author concluded that the only noticeable difference between those languages is the verbosity, as both of the languages are logical, machine and human readable. As the YAML syntax is more succinct and enables commenting, the author choses it to be the descriptive language for the SDL.

4 Scenario Definition Language structure

In this section the author first lays the groundwork for the SDL by specifying the communication method used for sending commands to the targeted hypervisors and VMs. After that, the author details the overall logic and functionality of the SDL while also describing the required actions taken by the exercise developers to use it.

4.1 SDL hypervisor communication method selection

When designing a hypervisor agnostic SDL, the author discovered two possible options, how to provision the SDL content onto the targeted hypervisors. The first option would be to write application programming interface (API) calls against desired hypervisors. The second option would be to communicate with the hypervisors through some configuration and automation tool that creates the API calls itself. The author chose the second option for a number of reasons.

To begin with, writing direct API calls to cover all the tasks required against different hypervisors is a lengthy process. This would mean that every new hypervisor integration needs a dedicated development cycle to cover all use cases that the SDL might require. Also, some bigger updates on the hypervisors may introduce changes to the API call syntax. In those situations, those already configured integrations must be revised and corrected to maintain their functionality.

This is where open-source configuration management tools step in. There is a variety of open-source tools available, as briefly described in the section 1.1. In this thesis, the author proposes to select one configuration management software and use that as a backbone for the SDL. In order to choose this said software, the author analyzed the most widely used provisioning and configuration management tools. In conclusion of the analysis, Red Hat Ansible [4] was the authors choice due to the following reasons.

First reason is that, Ansible uses YAML syntax by default. [48] This is desirable, as this means that the SDL can be defined in one markup language, bringing systemwide unity and clarity. Another helpful feature is that Ansible is agentless, meaning that all VM target configurations are done over Secure Shell Protocol (SSH) or Windows Remote Management (WinRM) and do not need any software preinstalled.

However, the two best features of Ansible are that it provides both provisioning and configuration functionality and that it supports a very wide range of hypervisors and cloud computing services, including, but not limited to VMware, OpenStack, Amazon Web Services, Google Cloud Platform, Microsoft Azure and Hyper-V. The full list of supported integrations can be found in their product description. [49] A survey by TechRepublic in 2019 [50] showed that Ansible is the most popular configuration tool used by respondents. Increasing use of the tool is a good indicator that assures the continued support of existing hypervisor modules. When a new release of a hypervisor platform changes some of its API calls, changes can be expected in the respective Ansible modules as well, thus eliminating the need to debug and rewrite ones' API calls. This guarantees that the SDL integrations with different hypervisors, once built, should be resistant to hypervisor updates and thus ensuring acceptable level of adapting to future changes.

4.2 Scenario definition

As discovered during literature review, there are many ways how CDX development can be approached. One approach method would be to design the SDL using high abstraction level artefacts. This would create a considerable amount of dependencies in respect to the number of artefacts, which must be handled. Another approach method would be to use low abstraction level artefacts, which can be used to interact directly with the target hypervisor or VM. In this thesis, the developed SDL will use a low abstraction level, VM centric approach when designing exercise scenarios. The author chose this approach in order to keep the VM definitions and hypervisor communication methods as short and straightforward as possible. With this approach, the author aims to achieve utmost clarity, which would encourage third parties to adapt the framework and ensure that scenario developments and new hypervisor integration processes (detailed in section 5.4) would be seamless and

successful. By using SDL, assuming that the correct hypervisor communication method has been developed, exercise developers would need only to define actual VM specific configuration from the state when the VM is already deployed and running on the target hypervisor. This will ensure that the development code will not contain any hypervisor specific information and can later be used to target other hypervisor platforms as well. At the time of writing, similar VM specific approaches are being used by the North Atlantic Treaty Organization (NATO) Cyber Range and NATO Cooperative Cyber Defence Centre of Excellence to develop and deploy their exercises. While the previously mentioned organizations also rely on augmented Ansible playbooks to deliver exercise-as-code, their exercises are developed against a single hypervisor platform and use custom-built toolset in conjunction with Ansible. The knowledge that low abstraction, VM specific approach is actively used to deploy some of the largest CDX-s to date [51], gives the author confidence that the chosen approach is applicable. The given SDL is designed using only open-source, community backed software which does not require any dedicated software from the underlying hypervisor.

The SDL is comprised of the following objects:

- Ansible files.

CDX provisioning and configuration is handled by Ansible, as it is the underlying tool for the SDL. The SDL Ansible files are grouped in three categories: core, conf and vm roles (section 4.3). These low abstraction level roles are required to define the actual commands which modify the target, whether it is a hypervisor or a VM, in a desired way.

- SDL files.

The SDL files are used to define the exercise scenario on a higher abstraction level. They rely on the existence of the previously mentioned Ansible files. One of those files, called *SDL_environment.yml* hosts a number of key-value pairs which are used to define exercise general parameters, such as hypervisor communication specifications, exercise domain name, default Domain Name System (DNS) servers, default hardware parameters, template information and more. *SDL_environment.yml* is covered in depth in section 4.2.1. The other file, called *SDL_exercise.yml* is used to define the deployable exercise information from the

VM perspective. In that file, VM specific information, such as hostnames, credentials, OS information can be defined. *SDL_exercise.yml* is covered in section 4.2.2.

- Template files.

In order to translate the contents of the SDL files for Ansible, dedicated template files will be used. In this thesis mako templating language [52] was chosen due to its ability to template nested variables and pass on regular Ansible variables without generating errors. Mako template usages are detailed in sections 4.5, 5.2 and 5.4.

From a technical standpoint, the SDL files provide an interface for the exercise developers to describe their exercise environment and setup by requiring them to insert values to the predefined keys. After that, a Python script, called *SDL_engine.py* (see Appendix 2: *SDL_engine.py*) must be executed. That script will populate those entered values to the specified destinations for Ansible to understand and operate with. In regular Ansible use-case, the users would be required to manually fill out multiple files, like the inventory file and a collection of different host and group variables. Additional complexity would be added to the previously mentioned actions when multiple hypervisor variables need to be defined. These SDL files, in conjunction with *SDL_engine.py* will make sure that the user has CDX oriented interface to work with and only defined target hypervisor variables will be used.

To use the SDL for exercise deployment, at least four conditions must be met:

- 1) the SDL files must be populated with environment and exercise information (section 4.2.1 and 4.2.2);
- 2) the Ansible VM role files must be defined (section 4.3.3);
- 3) OS templates must be defined (section 5.2);
- 4) the target hypervisor role must be present (section 5.3).

Once these conditions are met, the SDL can be used to deploy CDX into any defined hypervisor. All the code developed in this thesis is available at Github [53].

4.2.1 SDL environment definition

In the *SDL_environment.yml* file the user has to specify environment-based variables. Figure 4: *SDL_environment.yml*, populated with example values provides a visual overview of the first SDL file. To begin with, the first key, *environment* is required by the *SDL_engine.py*, to know, which hypervisor platform parameters it must process. Section 5.4 covers the details of integrating new hypervisors more in depth.

Then there are hypervisor access parameters, like the hypervisors Fully Qualified Domain Name (FQDN), username and password, which have enough permissions in the said environment. Defining specific minimal permission levels required in the target hypervisors will remain out of scope of this thesis. The account must have enough permissions to deploy the VM in the desired destination and modify its parameters. In this thesis system administrator level access is used. The keys for environment access parameters begin with prefix *env_*.

Then there are parameters with prefix *vs_* which define, where the VM will be placed within that environment. In the given thesis, there are only parameters for vSphere, since OpenNebula does not require such parameters to be defined. When targeting OpenNebula, or any other hypervisor besides vSphere, these *vs_* keys will be ignored and can thus be left empty. When integrating new hypervisors which require environment parameters, new keys must be created and a template file called *all.yml.tmpl* must be modified, to recognize the new keys. The template file has specific comments which guide the process.

Next off, there are global exercise parameters, like exercise name, which will be used as a prefix when generating a name for the VM and exercise domain. The latter will be used as a domain name for domain controllers and domain join operations, while also serving as DNS suffix for network connections. Also, primary exercise DNS servers must be specified, that act as default values for VM network configurations, unless some other DNS servers are specified in *SDL_exercise.yml*.

Following DNS, users must specify ansible deployer account. This is the account what Ansible uses to connect into the VM in order to perform post-deployment configurations. Ansible deployer account details are covered in section 5.2.

After ansible deployer specification, OS hardware defaults should be defined. These are the default virtual central processing unit (CPU) and random-access memory (RAM) allocations, when no exact VM level parameters are defined in the *SDL_exercise.yml*.

The final entries in the *SDL_environment.yml* are OS templates. They are laid out in a directory structure, which follow the OS family approach. For example, a template key for Microsoft (MS) Windows 10 template must reside under *os_windows*. An example of that be observed in Figure 4: *SDL_environment.yml*, populated with example values. The course of action for adding new OS templates is described in section 4.5.

If more OS-es are introduced to the environment, they just have to be referenced in the SDL in the same manner as current examples have been. Every new OS requires a unique key in the SDL and an accompanying name or ID from the environment.

```
environment: vsphere
env_hostname: 10.0.0.19
env_username: administrator@vsphere.local
env_password: Password

vs_datacenter: Datacenter
vs_exercise_name: SDL
vs_root_folder: VM_folder
vs_folder: VM_folder
vs_cluster: Cluster01
vs_datastore: Datastore
vs_resource_pool: RP_C01

exercise_name: sdl
ex_domain: sdl.local
ex_dns1: 10.0.1.11
ex_dns2: 10.0.1.12
```



```

ans_username: sdl
ans_password: Password.123

win_cpus: 2
win_ram: "16 GB"

lin_cpu: 2
lin_ram: "4 GB"

os_templates:
  os_linux:
    os_ubuntu: ubuntu_template
  os_windows:
    os_winserver: template_win_server_1809
    os_win10_21h1: template_win_10_21h1

```

Figure 4 : *SDL_environment.yml*, populated with example values

As displayed on Figure 4: *SDL_environment.yml*, populated with example values, the environment credentials are currently displayed in plaintext. This can be mitigated, by setting up Ansible vault [54]. In the ansible vault, two keys must be created, which will have the hypervisor account username and password as values. Those keys can then be used as variables instead of plaintext information.

For example, ansible-vault content can look like this:

- *environment_user*: Administrator@vsphere.local;
- *environment_pass*: Password.

In that case, the *env_username* key in *SDL_environment.yml* can have a value of “`{{ environment_user }}`” and the *env_password* can have a value of “`{{ environment_pass }}`”.

If the use of `ansible-vault` is desired, section 5.5 will give an example how to use the Ansible vault during deployment.

4.2.2 SDL exercise definition

The second SDL file the user has to modify is called *SDL_exercise.yml*. This file holds to-be-deployed VM specific information, which is mainly required by the hypervisor to assign correct physical parameters to the deployable VM, such as virtual CPU count. VM information is distinguished from each other by classifying them as separate dictionaries, called *vm1*, *vm2* etc. When adding new VMs, the same naming convention must be followed. For example, when 10 different VMs are to be deployed, VM entries must be specified from *vm1* to *vm10*. The order of the VMs does not matter.

Every to-be-deployed VM must have a corresponding VM section defined. In every section, there is a number of keys that need to be populated with values. The keys are illustrated in figure 5: An example VM definition in *SDL_exercise.yml*.

```
vm1:
  hostname: dc1
  parent: windows
  os: os_winserver
  network:
    name: virtualnet68
    ipv4: 10.0.68.68
    ipv4_gateway: 10.0.68.1
  dns_servers:
    dns1: 127.0.0.1
    dns2: 8.8.8.8
  cpu: 2
  ram: "16 GB"
  windows_user_accounts:
    username: Administrator
    password: Password.123
```

```
opennebula:
  new_network:
    name: virtualnet68
    ipv4: 10.0.68.11
    ipv4_gateway: 10.0.68.1
```

Figure 5: An example VM definition in *SDL_exercise.yml*

The first value that needs to be defined is the hostname of the VM. It must be noted that the name expected is in the form of hostname, not FQDN. The next two values are representing the OS family. In the key *parent*, the user must specify the general OS system type, like *Windows* or *Linux*. That key is used to by the *SDL_engine.py* to configure OS family wide parameters for the deployment. Next, the user must specify the exact OS type. The OS type value must correspond to the key that was defined in the *SDL_environment.yml*. For example, let's consider a VMware environment, where the user wishes to deploy an Ubuntu 20.04 machine, from a VM template called *ubuntu_template*. If in *SDL_environment.yml*, the user specified *os_ubuntu* key to have a value of *ubuntu_template*, then in the *SDL_exercise.yml*, the user must specify *os_ubuntu* as the value for the key *os*.

After specifying the OS, the user needs specify network parameters, such as network name, Internet Protocol version 4 (IPv4) address with its corresponding netmask, gateway and DNS. While IPv4 parameters are mandatory to be defined, DNS can be left unmodified. In that situation, default exercise DNS servers, specified in *SDL_environment.yml* will be used.

Following the network parameters, there are two optional parameters, called *cpu* and *ram*. Specifying those values will configure those values just for the given VM. If left unspecified, then the hardware default values form *SDL_environment.yml* will be used.

Next off, there is a selection that's mandatory only for MS Windows VMs, called *windows_user_accounts*. That value is required to fill if any domain specific operations will be performed, as those operations require a dedicated MS Windows account.

Finally, there are OpenNebula network specific parameters. Why OpenNebula requires and additional set of parameters, is detailed in section 5.2.2.

4.3 Roles

In Ansible, roles allow a user to develop standalone components that perform a series of defined tasks. In the SDL, the author has created three main role categories, called core, conf and vm.

Core roles (detailed in section 4.3.1) are a defined set of tasks which are required to deploy and prepare the VM for the upcoming VM specific configurations. Conf (detailed in section 4.3.2) is a collection of roles, which perform a specific list of configurations and are meant to be reusable. VM roles (detailed in section 4.3.3) contain VM specific commands, which can be unique, or previously mentioned reusable tasks.

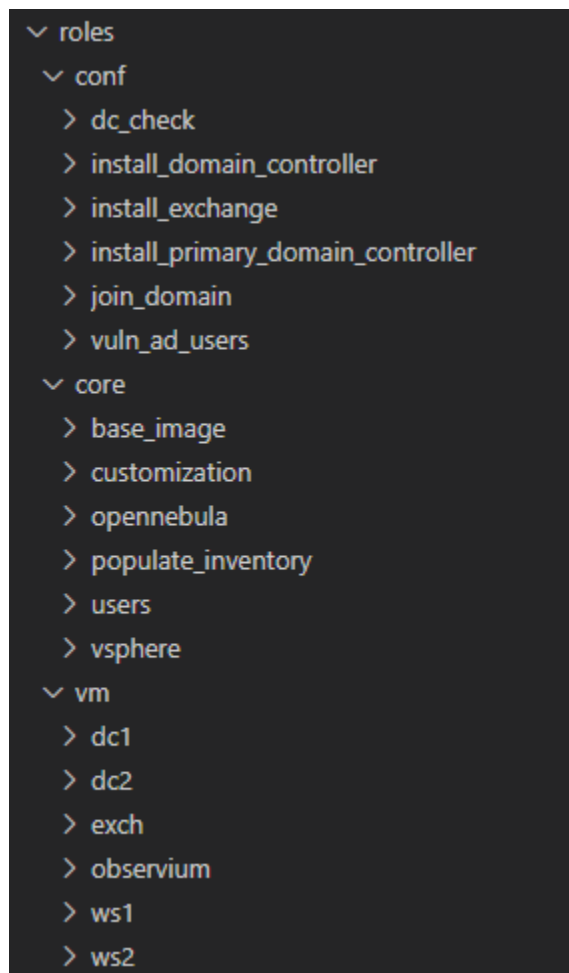


Figure 6: Ansible Roles

4.3.1 Core roles

Core roles are a set of tasks which perform the main operations required to provision and deploy a VM in the desired environment. These roles are always invoked in every ansible playbook execution, as illustrated in Appendix 1: Ansible playbook for vSphere deployment.

The first role that's invoked during a playbook execution, is *populate_inventory*. That purpose of that role is to make sure that correct host from the inventory with the right groups are targeted in the play. The inventory file is managed completely by the *SDL_engine.py*.

The second role that's invoked is the role of the respective target hypervisor. If the user wishes to deploy the VM into VMware vSphere, then the role will be *vsphere*, if the target hypervisor is OpenNebula, then the role will be *opennebula* etc. Creating new hypervisor roles is described in more detail in section 5.4. That role includes the tasks required to establish the connection with the hypervisor, cloning and deletion of the VM, configuration of networks and setting correct parameters for SSH connection. These hypervisor specific roles in principle setup a blank instance of the users to be configured VM. Hypervisor roles are more in depth detailed in section 5.3.

The third role that will be invoked is *base_image*. This role contains some basic configuration tasks for the targeted VM. For example, in MS Windows system, this role configures the time zone, makes sure that Remote Desktop Protocol (RDP) is enabled and changes the hostname into what's described in the SDL.

The fourth role invoked is *users*. In short, this role makes sure that the correct user accounts and profiles exist in the VM. In this proof of concept, only MS Windows OS makes use of this role.

The last role that's invoked is called *customization*. So far, all the core roles have been generic, only differentiating in terms of the target hypervisor and VMs underlying OS. The only purpose of the *customization* role is to call out host specific role from the *vm* directory, that will continue to configure the VM. For example, when deploying a domain controller VM called *dc1*, the customization role calls out the next set of tasks from *vm / dc1*. The *vm* category will be described in section 4.3.3.

4.3.2 Reusable configuration

When developing tasks for the target VM, some more generic actions can be filled out with variables, instead of specific host-based parameters and thus created into reusable tasks. For example, instead of writing a set of near identical tasks for the domain join operation for *ws1* and *ws2* VM, a predefined role from the tasks group can be invoked instead. This will take variables from the *SDL_environment.yml* and *SDL_exercise.yml* where needed, and modify the commands to suite the target VM.

Currently, the conf directory contains three primary task categories:

- *Install_* tasks, which install certain roles, like primary - and secondary domain controllers;
- *vuln_* tasks, which configure some sort of vulnerabilities;
- other uncategorized tasks, like domain join operations.

4.3.3 Virtual machine specifications

The *vm* category groups together VM specific configuration, based on the VMs desired outcome. For example, when deploying a MS Exchange server, then the tasks required to actually install and configure the service will be defined here. As installation of the MS Exchange server is a well-defined process, it can be written into a reusable function, as described in 4.3.2 and called out as a role. The user must then only specify actual MS Exchange server configuration.

When developing new VMs, basic Ansible directory structure applies [55]. In each directory, named after the desired VMs hostname, users may define the following standard Ansible directories:

- *tasks* – the only mandatory directory. In *tasks / main.yml* lays the main VM configuration;
- *defaults* – used to define the default variables within that role;
- *templates* – location for the templates;
- *files* – location for other files;
- *meta* – metadata for the role, including dependencies;

- handlers – dedicated tasks which are run only when notified;
- variables – other variables with higher precedence than defaults.

In this thesis, the author will not provide an in-depth analysis of the Ansible syntax possibilities. At the time of writing, Ansible contains over 750 modules [56], which target a wide array of different OS-es and resources. For a brief overview, a few commands from Observium VM configuration shall be examined. What is Observium and why it is used in this thesis is described in section 6 and the complete base configuration of Observium can be found in the respective GitHub repository¹, where the author has compiled the whole SDL configuration.

As the VM is running Ubuntu 20.04 OS, commands from the general Linux module will be used. The full path of file, in which the following configuration is located, is *roles / vm / observium / tasks / main.yml*.

```
- name: Download installer
  get_url:
    url: http://www.observium.org/observium-community-
latest.tar.gz
    dest: /opt/observium-community-latest.tar.gz
    mode: 755
```

Figure 7: Ansible *get_url* module example

The task portrayed on figure 7: Ansible *get_url* module example downloads an installer from the specified Uniform Resource Locator (URL), places into */opt/* directory and changes its permissions.

The following task, portrayed on figure 8: Ansible *mysql_db* module example uses a dedicated MySQL module to create a database for the observium.

¹ <https://github.com/xjan76/SDL/tree/main/roles/vm/observium>

```

- name: Create MariaDB database for observium
  mysql_db:
    name: observium
    login_user: root
    login_unix_socket: /var/run/mysqld/mysqld.sock
    login_password: Password.123
    state: present

```

Figure 8: Ansible *mysql_db* module example

As per Ansibles desired state configuration [57] the state *present* will make sure that the defined database exists; if not, it will be created. If the state would be *absent*, the command would make sure the database is deleted. These are just a couple of examples, how to comprise VM configuration out of Ansible modules. Alternately, if the use of Ansible modules is not desired, the whole VM configuration can be written into scripts, which can be copied to the target VM and executed there. Also, plain shell commands can be executed without the need to use ansible dedicated modules. For example, instead of installing packages on Ubuntu with a via dedicated apt module (Figure 9: Ansible *apt* module example), simple command can be entered (Figure 10: Ansible command example). This ensures a degree of liberty for the exercise developers. Those who wish, can make use of the ansible modules to configure the target VM. Those who do not wish to learn Ansible syntax, can just write direct commands or call out configuration scripts.

```

- name: Install packages
  apt:
    pkg:
      - libapache2-mod-php7.4
      - php7.4-cli
    state: present

```

Figure 9: Ansible *apt* module example

```

- name: Install packages
  command: "apt install libapache2-mod-php7.4 php7.4-cli"

```

Figure 10: Ansible *command* example

4.4 Virtual machine development lifecycle

VM development consists of two phases. In the first phase, the *SDL_exercise.yml* file must be populated with the new VM information, as detailed in section 4.2.2. In the second phase, a VM specific Ansible role must be developed. The VM specific ansible role creation is visualized in Figure 11: VM development lifecycle. Once the *SDL_exercise.yml* and *vm* roles are populated, the *SDL_engine.py* script must be executed. That script translates the *SDL_environment.yml* and *SDL_exercise.yml* content for the Ansible, consequently making sure that correct variables will be used. After that, ansible playbook can be executed to deploy the VM. Deployment procedure is described in detail in section 5.5.

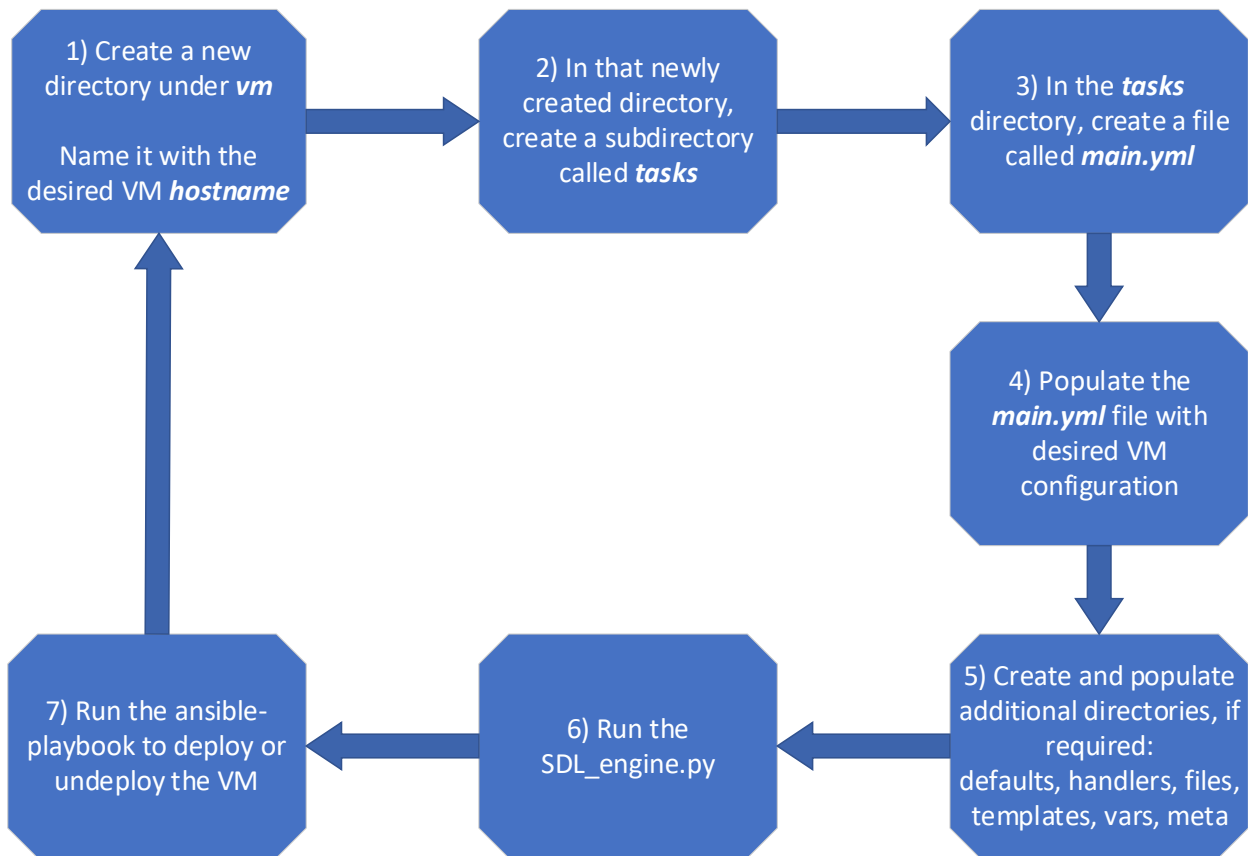


Figure 11: VM development lifecycle

4.5 Introducing new operating systems

When introducing new OS-es in the *SDL_environment.yml*, corresponding OS template files must be created. In order for Ansible to recognize the new OS information, the *SDL_engine.py* searches for OS template files and creates appropriate Ansible recognizable *group_vars* files out of it.

OS VM templates are defined in the end of the *SDL_environment.yml* file, and follow a hierarchical structure. First, it must be determined, whether the new template belongs to an existing OS family, or a new one must be created. As shown in Figure 12: OS templates in *SDL_environment.yml*, if for example a CentOS template needs to be added, it can be added under *os_linux*, on the same indentation as *os_ubuntu*. If, however a new type of OS is introduced, it must be specified right under *os_templates*.

```
os_templates:
  os_linux:
    os_ubuntu: ubuntu_template
  os_windows:
    os_winserver: template_win_server_1809
    os_win10_21h1: template_win_10_21h1
```

Figure 12: OS templates in *SDL_environment.yml*

The reasoning behind this approach is that on the OS family level, variables are defined for the all OS-es in that family. Under specific OS-es, unique variables, like template names will be defined.

Currently all OS templates must contain either direct or inherited values to the following keys: *ansible_user*, *ansible_password*, *customization_method*, *vm_template*. The list might grow, when introducing new functionalities to the SDL, which require dedicated host or group-based variables.

5 Hypervisor communication

In this section, the author will describe how the SDL will actually modify the target hypervisors. First of all, the author will describe the environment requirements for the SDL to work. Secondly, the author will give an overview, how to create functioning OS template images in both hypervisors. Thirdly, the author will explain the logic of the hypervisor targeting and give an overview, how to add new hypervisors to the SDL. Finally, the author will explain the deployment process.

5.1 Environment requirements

Before any hypervisor can be targeted, the environment must meet a number of basic requirements. Those requirements can be grouped to three main categories: networking, storage and compute resources.

5.1.1 Networking

Network configuration and deployment relies on assumption that the underlying infrastructure has basic networking pre-configured. This means that the hypervisor and its compute nodes have connectivity between each other. Due to the scope of this thesis, only two networks will be used. In order to verify the SDL being hypervisor agnostic, one network will be used in the vSphere environment and the other one in OpenNebula. Although both networks are present in both environments, this approach is taken in order to verify that the exercise can be deployed with different network parameters.

Unless a Software Defined Networking (SDN) solution, like VMware NSX-T [58] is used, then the creation of different Virtual Local Area Network (VLAN) backed networks require actions taken beyond the hypervisor infrastructure. This makes it difficult to handle within

the SDL, especially bearing in mind that different hypervisors are using different practices regarding network object definitions.

Network creation can partially be handled by the SDL. The requirement is that required VLANs are preconfigured and available to the hypervisor, thus only leaving the network object creation and desired VLAN association to the SDL.

When designing a scenario, VLAN IDs should not be hardcoded anywhere. This will allow the scenario to be hosted in a different environment where a different set of VLANs are possibly available. The SDL must also provide a choice, whether to create new network objects either with or without desired VLAN associations or use existing objects.

5.1.2 Storage

The storage configuration also relies on assumption that underlying infrastructure has some form of storage attached and configured. The storage can be implemented either as local storage on physical hosts, local storage of virtualization platform like OpenNebula Sunstone, dedicated storage platform like Dell XtremIO [59] or Hyperconverged Infrastructure (HCI) solution, like VMware vSAN [60].

Due to storages being considered a more static part of a virtual environment, the SDL, at least in its proof of concept state, will not focus on creating new datastores for exercises. The reason for this is that datastores are often managed outside of hypervisor layer. If datastore creation would be seen necessary, then additional toolset should be configured which handles datastore management. When designing an exercise, the targeted storage solution for exercise deployment must be defined in the *SDL_environment.yml*.

5.1.3 Compute resources

Compute resources provide computational power and memory to the VMs which reside on physical hosts. In different hypervisors, compute resource management is handled differently. In VMware vSphere environment, resource management is hierarchical. Clusters are used to group physical hosts. When a host is added to a Cluster, the hosts resources become a part of the Clusters resources. The Cluster manages the resources of all the hosts within it.

Resource pools in conjunction with VMware vSphere Distributed Resource Scheduler (DRS) distribute the VMs between different hosts to ensure that no host is overburdened either via CPU or RAM usage. It can be thought of as an active load balancer within the cluster. In addition, resource pools can function as an additional layer of access control. The SDL will require either a single host do be defined as the target for the VM, or a resource pool, which will choose the best suiting host itself. The compute resource hierarchy is visually displayed in Figure 13: vSphere appliance. Although in this proof of concept, only a single ESXi host is in the cluster, resource pool is still created to test the functionality.

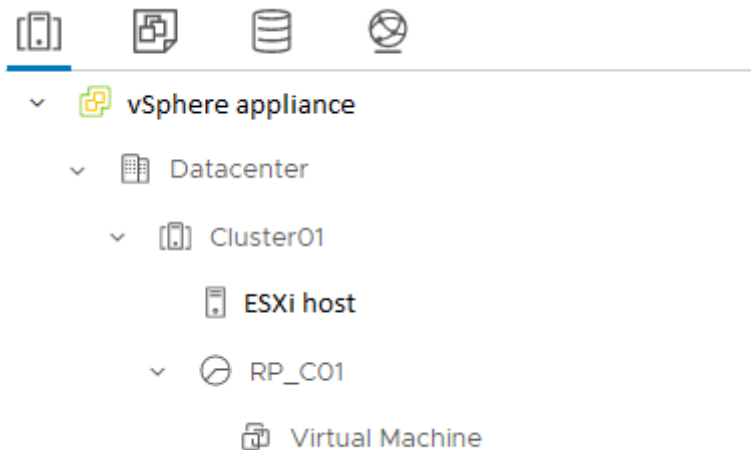


Figure 13: Compute resources in vSphere appliance

In OpenNebula environment, there is no similar concept of resource pooling, as displayed in Figure 14: Compute resources in OpenNebula appliance. Besides that, the concept of hosts and clusters in OpenNebula remain similar to VMware vSphere. When targeting OpenNebula, the deployed VM will by default be instantiated on the same host where the template is located. If that is not desired, a specific host must be specified during the deployment.

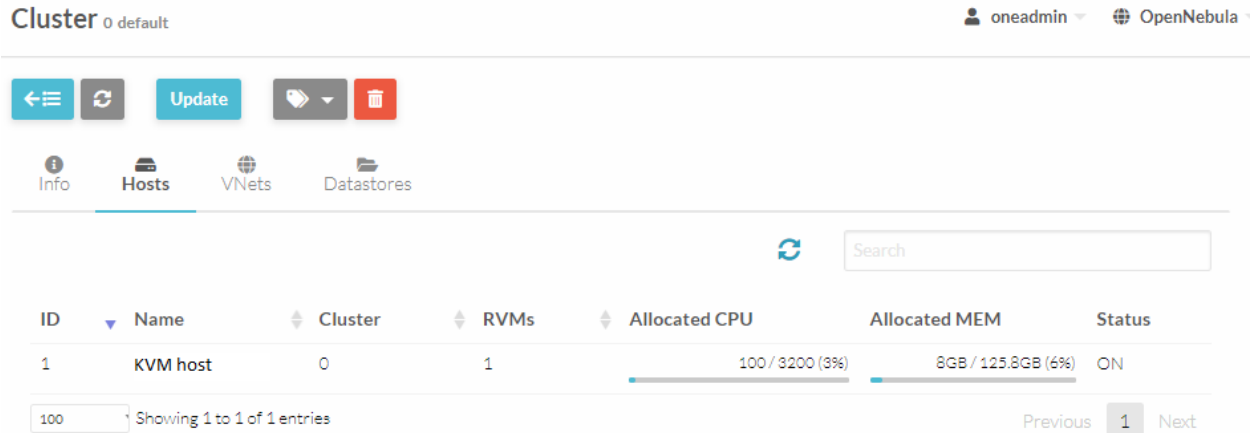


Figure 14: Compute resources in OpenNebula appliance

5.2 Templates

The deployment of virtual machines will be based on pre-built templates since installing fresh OS from disk image file for every VM is very time consuming, especially considering Microsoft products. In this thesis, the author created a template image for each OS:

- A MS Windows Server 1809 template
- A MS Windows 10 version 21h1 template
- A GNU/Linux Ubuntu 20.04 server template

Depending on the environment, templates can be created using some automation software, like HashiCorp Packer [61], or installed manually. In this proof of concept, the author used both methods, to describe the workflows. Both of these methods will be detailed in the following sections.

In the following two chapters, overviews will be given on how the templates should be prepared. As the MS Windows OS requires much more pre-configuration than GNU/Linux, the author has also created a PowerShell script for preparing MS Windows VM. The script, called *prepare_windows_templates.ps1* can be found in GitHub². In that script, desired username

² https://github.com/xjan76/SDL/blob/main/prepare_windows_templates.ps1

and password must be inserted. The following points in the upcoming sections just describe the actions taken within the script.

5.2.1 VMware virtual machine templates

From the SDL perspective, it does not matter, how much CPU, RAM, and disk space is allocated to the VM template. CPU and RAM allocations will be modified during the VM deployment, as specified in the SDL. When choosing disk size, exercise size and datastore capacity must be considered. In the current proof of concept, exercise defined in *SDL_exercise.yml* can only be deployed on one specific datastore, which is defined in *SDL_environment.yml*. This means that available space in target datastore must be evaluated, before initiating deployment to ensure that defined VMs will. If deployment on multiple datastores is required, then the *datastore* key in *SDL_environment.yml* must be changed between deployments and *SDL_engine.py* must run. This will ensure that the next deployment will target a different datastore.

In VMware vSphere environment, *Thin Provisioning* is the recommended VM storage policy type, as it will only use as much disk space as the OS actually requires, not the whole fixed amount, which is defined during VM template creation. Regarding networking, a vSphere *portgroup* with DHCP service enabled should be used for template, as the vSphere Ansible module can detect DHCP addresses thanks to VMware Tools [62] module. This will allow multiple cloning tasks to be run in parallel, as each powered-on template clone, which has not yet entered IPv4 configuration phase, can receive a unique IPv4 address. This is not the case with OpenNebula, which will be discussed in the next section.

In this example, when creating MS Windows templates by hand, the following points should be addressed:

1. User accounts

All MS Windows templates should have two local administrator accounts. The first one is the account, which is referenced in the *SDL_exercise.yml*, and the second one, which is referenced in the *SDL_environment.yml*. The first account, called Administrator in this example, is the account which shall be used for tasks, like domain join operations etc. From the exercise perspective, that account can be considered as an in-game account. The

second account, called *sdl* is the account, which is used by the Ansible connection to perform configuration steps. That account can be considered as an external account. Meaning that for example, if in any logs the *sdl* account is detected, it is clearly visible, that those operations are performed by the exercise developers and not by any in-game party.

2. Software

First of all, MS Windows package manager Chocolatey [63] should be installed, as this simplifies software management on MS Windows OS-es. Secondly, since all MS Windows Administration is done over SSH, OpenSSH [64] must be installed and configured.

Also, previously mentioned VMware Tools installation is a must, as the Ansible vSphere module depends on it during DHCP address detection.

3. Additional services

If desired, incoming Echo Requests can be opened from the firewall and RDP can be enabled. Those features might prove useful but are not required.

Another thing to note is that in the SDL, templates in vSphere environment are referred to by their name. This means that when the templates are created, the names assigned must be the same names as the values to the keys in the *SDL_environment.yml* file.

As mentioned in chapter 5.2, VM template creation can also be automated. In this thesis the author used HashiCorp Packer to automate MS Windows templates creation in vSphere environment. As using Packer is an optional feature, its' installation and configuration are not handled by the *install_requirements.sh* script, detailed in section 5.5.

Once Packer is installed and setup as detailed the guidelines in its official documentation [65], dedicated directories should be created for every template. With MS Windows OS use-case, those directories must be populated with two files: *Autounattend.xml* and a configuration file, written either in JSON or HashiCorp Language (HCL). The authors'

provided examples³ can be used once the variables in the JSON configuration files are changed to match the target environment.

Regarding Ubuntu Server 20.04, if the correct account – the one described in *SDL_environment.yml*, is created and SSH is enabled during the installation, then the only thing left to check is that VMware Tools would be installed.

5.2.2 OpenNebula virtual machine templates

In OpenNebula, the process of VM template creation is similar to vSphere, but with some caveats. To the authors knowledge, OpenNebula does not have management module similar to VMware Tools. Because of this, templates must be created with static IPv4 addresses. That static address has to be defined in the *network* parameters in *SDL_exercise.yml*. This ensures that Ansible OpenNebula role can locate the target VM, connect with it, and change its IPv4 address to the one specified under *new_network* parameter. This applies to both MS Windows and GNU/Linux OS. Otherwise, the template installations follow the same procedure as in vSphere.

Another difference is that in OpenNebula, VM templates and virtual networks can be referred to by their template ID and network ID, respectively. This is because OpenNebula allows to create multiple templates and networks with the same name. In this thesis, the author still uses template and network names in the files for the sake clarity, but when addressing future, full scale exercises, then the use of IDs instead of names could be considered. In that case, in the *SDL_environment.yml*, template keys must have template IDs, instead of names as their values and in the *SDL_exercise.yml*, OpenNebula network ID must also be specified.

Regarding MS Windows OS templates, as there is no way to change the VM Security Identifier SID during the cloning process, the VMs must be generalized [66] before they can be considered finished. In order for the VM to boot up automatically, the author prepared an *unattend.xml* file⁴, which is placed on the templates C: drive. When the template configuration has finished, the machine will be generalized with the following command: `sysprep /oobe /generalize /shutdown /unattend:c:\unattend.xml`. The command should be run

³ https://github.com/xjan76/SDL/tree/main/packer_vmware

⁴ https://github.com/xjan76/SDL/blob/main/packer_vmware/unattend.xml

in the `c:\windows\system32\sysprep` folder. This will make sure that once the vm will be booted up after the cloning, it will have correct parameters automatically set.

5.3 Targeting hypervisors

For each desired hypervisor, a dedicated hypervisor role is required to be developed. In this proof of concept, VMware vSphere and OpenNebula roles have been created (refer to sections 5.3.1 and 5.3.2 respectively).

5.3.1 VMware vSphere

The vSphere role consists of six main tasks, which are invoked in the order that's specified in `vsphere/tasks/main.yml`. The first tasks, called `vmware_guest.yml` sets correct IPv4 parameters for `vmware_guest` module, which will be used later in the VM cloning process.

The second task, named `undeploy.yml` will be invoked only when `deploy_mode` parameter is set to `undeploy` or `redeploy` in the ansible-playbook command. This role powers off the VM and deletes it from the datastore.

The third task, called `portgroups.yml` checks, whether the correct portgroup exists for the deployment and if not, creates it. In this proof of concept, the basic functionality for the task is included. For more complex use-cases, additional development might be required in order to meet the expected functionality. This is an optional task, which can be commented out in the `vsphere/tasks/main.yml` task, if portgroup creation is not expected from the SDL.

The fourth task is called `deploy.yml`, which deals with the VM deployment. It first checks the environment for the VM and if it doesn't exist, deploys and powers it on. Because MS Windows OS network will already be configured during the deployment process, then a MS Windows IPv4 validation block is added to the end of the task. The reason why customization is done for MS Windows during the cloning process is to change the VM Security Identifier (SID), in order to prevent conflicts in the domain join operations.

The fifth task, called `network.yml`, configures – and in MS Windows OS case, validates – the correct IPv4 address for the VM, as defined in `SDL_exercise.yml`.

The final task is called *connection.yml*, which makes sure that Ansible can establish SSH connection the newly assigned IPv4 address. Once the SSH connection is established, the vSphere role finishes and the playbook execution continues with the next core configurations, yet those are already hypervisor agnostic.

5.3.2 OpenNebula

What differentiates OpenNebula from vSphere in regards of VM deployment, is the concept of instantiation. In vSphere, VMs are cloned from a template and after cloning, the VMs are no longer associated with the template. In OpenNebula, the VMs are different instances of the base template. For example, after instantiating two domain controllers from the MS Windows server template, those two machines are just a different instance of the said template.

Instantiation also means, that disk image file mounting must be done on the template level. This also sets OpenNebula logic apart from vSphere, as the reusable functions in *roles/tasks* directory, which need to mount a dedicated disk image file, for example like MS Exchange server installation, require a dedicated template. This is something that need to be considered when integrating new hypervisors into the SDL.

The OpenNebula role consists of five main tasks, which are invoked in the order specified in *roles/core/opennebula/tasks/main.yml*. The first task invoked, is *undeploy.yml*, which works the same way as described in the previous section.

The second task invoked is named *deploy.yml*. This task will instantiate the template with the parameters described in the *SDL_exercise.yml*. At the time of writing, Ansible OpenNebula *one_vm* module does not seem to have a method of querying existing instance information. The author resolved that issue with a ping check. This means that before any new instantiation, the VMs IPv4 address, as defined in *SDL_exercise.yml* will be pinged and if an answer is received, the instantiation task is skipped and the playbook execution continues with the configuration tasks.

The third task, named *connection.yml* creates an initial connection to the freshly initiated VM. As mentioned in section 5.2.2, since OpenNebula does not have its own management

tools, a static IPv4 has to be set on the templates, which can be used as an initial entry point to the VM. The task finishes once an SSH connection has been established with the VM.

The fourth task, called *network.yml* sets the correct network parameters to the VM, as described in the *SDL_exercise.yml*. As this is done over SSH, and not over management tools, the connection between Ansible and VM will momentarily break, resulting in an *unreachable* error. In this proof of concept, the error is handled with *ignore_unreachable* key for that specific task. This ensures that even though connection breaks, the playbook execution will continue and the next task will establish a connection with new network parameters.

The last task called is named *new_connection.yml*. As the previous task changed the network parameters of the machine, this task establishes a new SSH connection. With that task, the OpenNebula role finishes and the playbook execution moves on with the rest of the configurations.

5.4 Integrating new hypervisors

Integrating new hypervisors consists of two main steps. First, a hypervisor specific subdirectory must be created into the *core* directory. The name of the subdirectory must be noted for the second step. Within that directory, there are a number of tasks which must be implemented in order to maintain the functionality of the SDL. The easiest way to proceed is to take either the existing vSphere or OpenNebula subdirectory as a template and modify the tasks with the correct hypervisor modules and commands. There must be tasks, which deal with VM deployments and undeployments, depending on the value of the key *deploy_mode*, which is set during playbook execution. There must also be a task, which configures the correct network parameters for the VM, and a task, which establishes an SSH connection. Once those primary functions are operational, the new environment can be considered integrated.

Secondly, a new playbook must be created, which calls out the correct hypervisor role. Again, this can easily be done by copying the content of either *vsphere.yml* or *opennebula.yml* to a new file and then just changing the hypervisor role. The hypervisor role is also highlighted on the Figure 15: hypervisor role in *vsphere.yml* playbook. If the said VMware vSphere

playbook is taken as a template, then the name of the subdirectory, chosen in the first step, must be inserted in place of *vsphere*.

```
roles:
- core/vsphere # hypervisor specific role
- core/base_image
- core/users
- core/customization
```

Figure 15: Hypervisor role in *vsphere.yml* playbook

Finally, a mako template, residing at *group_vars / all.yml.tmpl* must be edited. If the new hypervisor has any module defaults or any other unique variables, they must be added to the template in a similar manner as currently vSphere and OpenNebula have. The template file has specific comments which help guiding the process. Also, if the new hypervisor requires any new packages, they should be added to the *install-requirements.sh* script, which is detailed in the next section.

5.5 Deployment

Once the *SDL_environment.yml* and *SDL_exercise.yml* files have been populated, VM configurations developed and the *SDL_engine.py* executed, the VMs are ready to be deployed. The deployment will be executed from the exercise developer's personal computer.

Before the actual deployment can commence, a couple of requirements must be met. First of all, the deployment requires a terminal with GNU bash shell [67]. In most GNU/Linux OS-es, bash is the default shell. In Apples' macOS, if bash is not a default shell, it can be changed into via `chsh -s /bin/bash` command. Regarding MS Windows OS, a Windows Subsystem for Linux (WSL) is required. By default, Ubuntu distribution will be installed, but this can be changed if desired. Official Microsoft documentation [68] can be referred to for installation guides and best practices, if required.

Secondly, the deployer must verify that Python 3.9 is the chosen python interpreter. Once that is verified, the *install-requirements.sh* script, found in GitHub⁵, should be run. It will install all the required packages and collections which are required for the SDL. Currently, in this proof of concept, only Debian based OS-es are handled by the script.

The ansible-playbook command is built up as follows:

```
Ansible-playbook [hypervisor specific playbook name] -e=deploy_mode=[deploy_mode] -e=machine=[VM name]
```

As introduced in section 5.3.1 the `deploy_mode` accepts three different values:

- Deploy – the VM will be deployed. In case of vSphere environment, if the VM already exists, the deployment part will be skipped and the playbook execution will continue with the configuration tasks. OpenNebula, as mentioned in section 5.3.2 will deploy a new instance regardless whether an instance is already present or not.
- Undeploy – the VM will be powered off and deleted.
- Redeploy – this role first executes the task undeploy and then deploy.

For example, to deploy a VM named *dc1* in the OpenNebula environment, the command would be: `Ansible-playbook opennebula.yml -e=deploy_mode=deploy -e=machine=dc1`

In another example, to redeploy – meaning to undeploy and then to deploy two VMs named *ws1* and *ws2* in vSphere, the command would be: `Ansible-playbook vsphere.yml -e=deploy_mode=redeploy -e=machine=ws1,ws2`

As can be observed from the vSphere example, multiple VMs can be deployed at the same time. Those tasks will then run concurrently and that's something to bear in mind, when VMs have dependencies. Deploying primary and secondary domain controller concurrently will most likely end in failure for the secondary domain controller, as the primary domain controller might not be properly configured on time.

In terms of security, Ansible vault can be used to encrypt plaintext credentials. If the use of Ansible vault is desired, then it must be invoked during the deployment. For example, if

⁵ <https://github.com/xjan76/SDL/blob/main/install-requirements.sh>

ansible vault was named *environment*, then an example playbook command should look like the following:

```
Ansible-playbook opennebula.yml -e=@environment --ask-vault-pass -  
e=deploy_mode=deploy -e=machine=dc1
```

6 Proof of concept implementation

In this section, the author will describe an example scenario and deploy it to two different hypervisor platforms. First, the author will introduce the deployable exercise layout and describe each scenario entity. After that, the author will deploy the scenario into different hypervisor platforms and evaluate the process.

In this proof of concept, a small Microsoft Active Directory based exercise environment will be developed and deployed on VMware vSphere and OpenNebula platforms. Figure 16: Network map gives a visual representation of the deployable VMs on the target hypervisors. As can be seen from the same figure, routing will be handled by the firewall outside of the deployable exercise. This is due to the fact that basic network connectivity on the hypervisor platform is a prerequisite for using the SDL, as described in section 5.1.1.

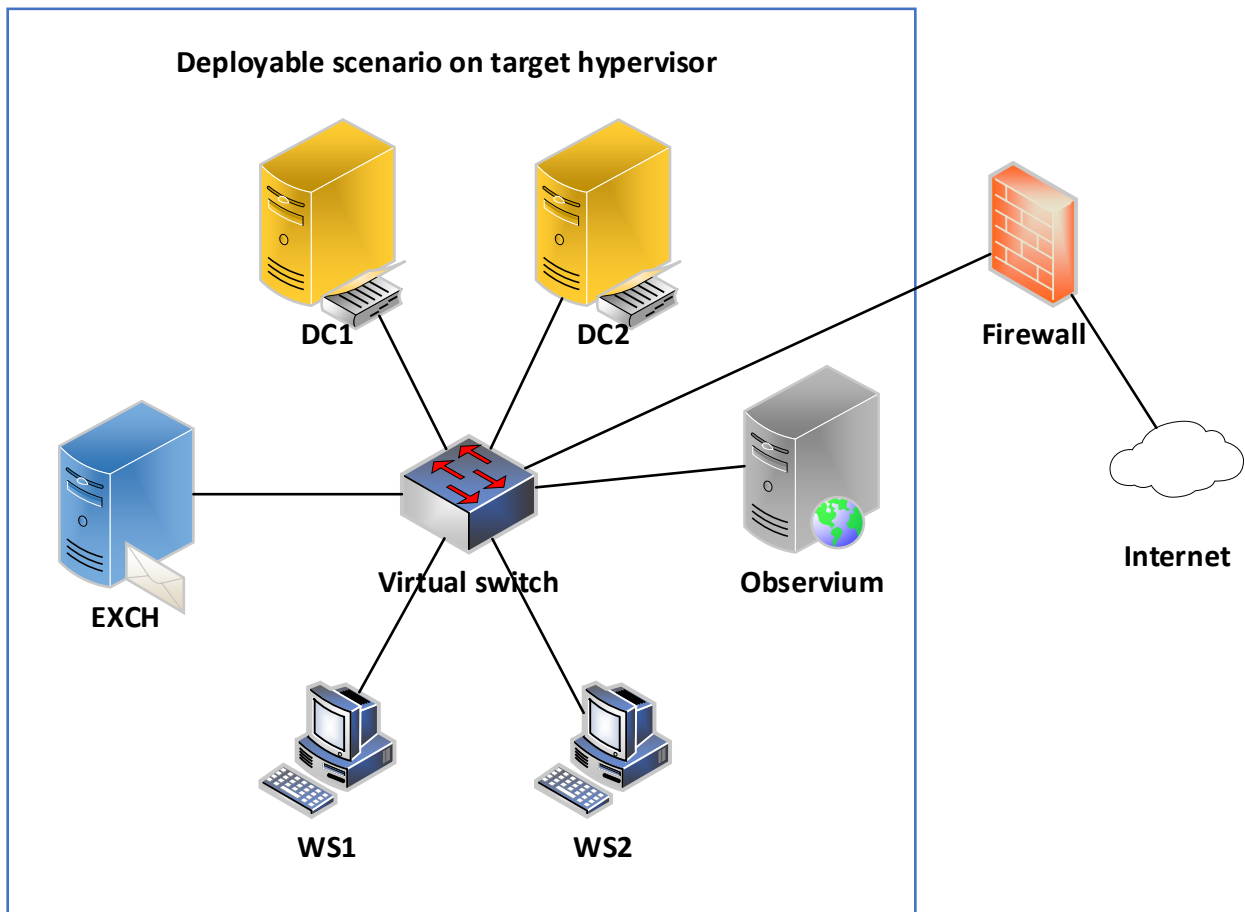


Figure 16: Network map

To begin with, two domain controllers – *dc1*, a primary domain controller and *dc2*, a backup domain controller will be deployed and configured. Besides base configuration, some user accounts will also be created and vulnerabilities added⁶. Appendix 3: vSphere deployment example and Appendix 4: OpenNebula deployment example represent deployment logs for *dc1* into respective environments.

A MS Exchange server, named *exch*, will be added to the Microsoft domain environment. Base configuration⁷ will be applied to the MS Exchange server, meaning that it can be used to send out emails.

⁶ <https://github.com/xjan76/SDL/blob/main/roles/vm/dc2/tasks/main.yml>

⁷ <https://github.com/xjan76/SDL/blob/main/roles/vm/exch/templates/Configure-Exchange.ps1>

Aside from MS Windows Servers, two MS Windows 10 workstations, called *ws1* and *ws2* will be deployed, configured and domain joined. These workstations will act as office workstations for the previously mentioned user accounts.

Alongside MS Windows OS, an Observium [69] service will be deployed on a GNU/Linux Ubuntu Server 20.04. This is to test and verify a non-Windows VM deployment and configuration. Observium was chosen due to its popularity and the need to handle a variety of different steps during installation⁸. Besides basic server setup, Lightweight Directory Access Protocol (LDAP) will be configured to allow specific Active Directory users⁹ to log into the service as administrators.

This proof of concept implementation can be considered successful, once the defined exercise is successfully deployed onto two different hypervisor environments. The implementation process will consist of four main phases.

In the first phase, six aforementioned VMs with their respective roles will be described in the *roles / vm* directory.

In the second phase, *SDL_environment.yml* and *SDL_exercise.yml* files will first be populated with VMware vSphere platform information. The *SDL_engine.py* will be executed and Ansible playbook deployment commands will be executed in the order described in section 6.1.

Once the VM deployments are finished in the vSphere environment, the *SDL_environment.yml* and *SDL_exercise.yml* files will be populated with OpenNebula platform information. While the *SDL_environment.yml* file will see considerable changes, only network information shall be modified in the *SDL_exercise.yml*. The *SDL_engine.py* will be once again executed and Ansible playbook deployment commands will be executed in the order described in section 6.2.

Finally, once the VM deployments are finished in OpenNebula platform, both hypervisor instances will be examined and the VM functionalities will be verified. The examination process is visual, meaning that deployment logs will be checked for errors and deployed VMs

⁸ <https://github.com/xjan76/SDL/blob/main/roles/vm/observium/tasks/main.yml>

⁹ <https://github.com/xjan76/SDL/blob/main/roles/vm/observium/templates/config.php.j2#L46>

will be inspected on the hypervisor, to verify that they have correct physical parameters, like CPU, RAM and network adapter configured. VM functionality will be verified by checking, whether the VM configurations, which were described under *roles / vm* and also in *SDL_environment.yml* have been implemented. If the instances are working identically, the proof of concept implementation can be considered successful.

6.1 VMware vSphere

The scenario was deployed into VMware vSphere environment in the following order:

- 1) *dc1* – Primary domain controller;
- 2) *dc2* – Secondary domain controller;
- 3) *exch* – MS Exchange server;
- 4) *observium* – Observium server;
- 5) *ws1* and *ws2* – domain joined MS Windows 10 workstations.

The first two VMs – *dc1* and *dc2* had to be deployed one at a time. This was because domain creation within *dc1* had to be finished before secondary domain controller could be joined to the freshly created domain. As both of these domain controllers serve as DNS servers for the rest of the VMs, *dc2* deployment had to be finished before other VMs could be deployed.

Although MS Exchange, Observium and both workstations could have all been deployed together, the author decided to only deploy the workstations in parallel. The deployment logs for the parallel deployment can be found in Appendix 5: vSphere dual deployment example. The reasoning for that choice was that firstly, deployment times varied a lot. An average MS Exchange Server deployment took approximately 70 minutes, while workstation deployments were finished in approximately 10 minutes and Observium server deployment averaged around 20 minutes. While there is nothing wrong with different deployment times, it does make the deployment log inspection harder. If deployment log clarity is a priority, it would be possible to open multiple bash shells and execute the deployment commands at the same time in separate shells. This would enable faster deployment process while keeping the logs separate.

Secondly, since the proof of concept environment only consisted of one VMware ESXi host, which had 50 GHz of CPU and 64 GB of RAM available, deploying four VMs in parallel, from which three were MS Windows OS-es, used up nearly all the resources during initial VM configuration phase. While testing this approach, the author noticed that deployment times increased approximately two times. When deploying the VMs in the order specified in the beginning of the section, no anomalies were detected and the deployment of the scenario vSphere environment went smoothly.

The SDL makes use of VMware vSphere management tools, called VMware Tools [70] in the first parts of the deployment via Ansibles *vmware_guest* module. VMware Tools are used to gather the deployable VMs IPv4 address, assigned by DHCP server, to create an SSH connection. This enables multiple parallel deployments from one template, as every cloned VM will receive unique IPv4 address during its initial bootup. The existence and use of such management tools are highly desired for larger, multi team scenario deployments as it would enable the same VM to be deployed and configured for multiple teams at the same time.

<input type="checkbox"/>	Name	↑	State	Used Space	Host CPU	Host Mem	DNS Name
<input type="checkbox"/>	sdl_dc1		Powered On	25.54 GB	167 MHz	5.58 GB	dc1.thesis.demo
<input type="checkbox"/>	sdl_dc2		Powered On	25.51 GB	125 MHz	5.52 GB	dc2.thesis.demo
<input type="checkbox"/>	sdl_exch		Powered On	32.55 GB	4.9 GHz	16.09 GB	exch.thesis.demo
<input type="checkbox"/>	sdl_observium		Powered On	29.08 GB	62 MHz	3.61 GB	observium
<input type="checkbox"/>	sdl_ws1		Powered On	31.11 GB	146 MHz	5.49 GB	ws1.thesis.demo
<input type="checkbox"/>	sdl_ws2		Powered On	31.83 GB	146 MHz	5.16 GB	ws2.thesis.demo

Figure 17: Deployed exercise in VMware vSphere

As can be seen on Figure 17: Deployed exercise in VMware vSphere, six machines are up and running in the VMware vSphere environment. All the machines are reachable via the IPv4 addresses described in the *SDL_exercise.yml*, MS Windows VMs have been joined to

the domain, that was specified in the *SDL_environment.yml* as *thesis.demo* and Observium service can be accessed by the domain account with administrative privileges, that was specified in *dc1* tasks¹⁰. The deployment log of *sdl_dc1* VM can be found in Appendix 3: vSphere deployment example. The deployment log also displays timestamps to indicate, how long each deployment step took time. As can be verified from the deployment logs, *dc1* was deployed and configured in 21 minutes. All the VM configurations for the example deployment have been left unchanged and are accessible in the GitHub repository¹¹.

6.2 OpenNebula

The scenario was deployed into VMware vSphere environment in the following order:

- 1) *dc1* – Primary domain controller;
- 2) *dc2* – Secondary domain controller;
- 3) *exch* – MS Exchange server;
- 4) *observium* – Observium server;
- 5) *ws1* - domain joined MS Windows 10 workstation;
- 6) *ws2* – domain joined MS Windows 10 workstation.

Deploying the exercise to OpenNebula platform followed mostly the same order as described in previous VMware vSphere section but was slightly more challenging. The author noticed that when redeploying a VM, the deployment task can randomly fail. This would happen approximately every 10th redeployment process. The hypervisor logs indicated that although the VM instance was deleted, the instance information would not be completely removed from the database and during a new instantiation, OpenNebula could not properly create a new database entry. The problem seemed to be caused by too short timeframe between instance deletion and a new instantiation. A workaround for that was to undeploy the instance, wait for approximately 20 seconds and then deploy again. This way the deployments succeeded every time.

¹⁰ <https://github.com/xjan76/SDL/tree/main/roles/vm/dc1>

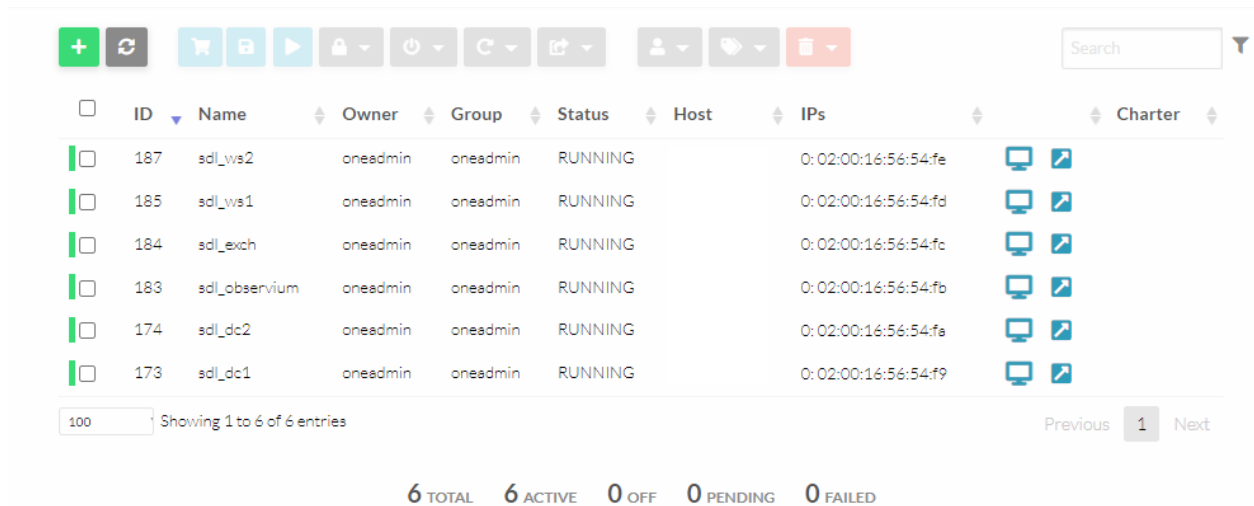
¹¹ <https://github.com/xjan76/SDL/tree/main/roles/vm>

Another challenge with deployment to OpenNebula was that since there are no hypervisor management tools, VM templates must use static IPv4 address. This is due to the fact that SDL needs to establish SSH connection to the freshly cloned instance, but as there are no management tools to discover the DHCP address, a static IPv4 address must be configured in the template and described in the SDL. This resulted in a situation that any single template could not be used for cloning more than one instance at the time, as it would result in an IP conflict. For example, it was possible to deploy *observium* and *ws1* at the same time, but it was not possible to instantiate *ws1* and *ws2* in parallel, as they both used the same MS Windows 10 template. The author did not find a feasible workaround for that issue.

Ansible OpenNebula modules are considerably less mature compared to VMware vSphere modules. At the time of writing, *one_vm* module, used for VM instantiation does not have a good way for looking up an existing instance information. The problem which arises from such a situation is that when existing instance information cannot be queried, every VM instantiation must be started from the beginning. This limits the possibility of performing minor changes to already deployed VMs. Although the modules *count_attribute* key can be used to delete an instance by its name attribute, the only way the author managed to determine whether an instance exist was to perform a ping check. While the ping check approach did resolve the issue in this proof of concept, some alternative method must be developed if ping cannot be enabled in the exercise network. Another option would be to redeploy the instance during every playbook execution. This option works fine, but is considerably more time consuming.

As can be verified from the deployment log timestamps (Appendix 4: OpenNebula deployment example), *dc1* deployment to OpenNebula platform took 24 minutes. The author noticed similar tendencies with all VM deployments; OpenNebula deployments were always approximately 15% slower compared to VMware vSphere. This was most likely caused by the compute resource differences. While the underlying KVM host in OpenNebula was equipped with Intel Xeon E5-2630 processor, the VMware ESXi host possessed Intel Xeon Gold 6230 processor, which was more capable. The author did not have an opportunity to test the deployment speeds on equal hardware, but a 15% speed difference due to hardware

difference seems plausible, especially as the deployment speed difference occurred mostly during deployment tasks.



The screenshot displays the OpenNebula dashboard interface. At the top, there is a toolbar with various icons for actions like adding, refreshing, and deleting. Below the toolbar is a search bar. The main content is a table with columns for ID, Name, Owner, Group, Status, Host, IPs, and Charter. The table lists six VMs, all with a status of 'RUNNING'. Below the table, there is a pagination control showing 'Showing 1 to 6 of 6 entries' and a summary bar indicating '6 TOTAL', '6 ACTIVE', '0 OFF', '0 PENDING', and '0 FAILED'.













ID	Name	Owner	Group	Status	Host	IPs	Charter
187	sdl_ws2	oneadmin	oneadmin	RUNNING		0:02:00:16:56:54:fe	 
185	sdl_ws1	oneadmin	oneadmin	RUNNING		0:02:00:16:56:54:fd	 
184	sdl_exch	oneadmin	oneadmin	RUNNING		0:02:00:16:56:54:fc	 
183	sdl_observium	oneadmin	oneadmin	RUNNING		0:02:00:16:56:54:fb	 
174	sdl_dc2	oneadmin	oneadmin	RUNNING		0:02:00:16:56:54:fa	 
173	sdl_dc1	oneadmin	oneadmin	RUNNING		0:02:00:16:56:54:f9	 

Figure 18: Deployed exercise in OpenNebula

As can be seen on Figure 18: Deployed exercise in OpenNebula, the same six machines are running in the OpenNebula environment. Although OpenNebula dashboard does not display deployed VMs FQDN, it can be verified from Appendix 4: OpenNebula deployment example, that OpenNebula role successfully deploys the VM and the playbook moves on to hypervisor agnostic customization roles. The only difference between VMware vSphere and OpenNebula exercise environments is that they are using a different VLANs, which were specified in the *SDL_exercise.yml*. Other than that, the instances are identical, as the VM configuration roles are not tied to hypervisor specific roles.

7 Discussion

The author has compiled all the code and stored it for public use and validation under Massachusetts Institute of Technology (MIT) license on GitHub page¹². Currently, this proof of concept covers the primary functionality of a CDX development and deployment. Existing deployment roles could be improved to handle VM team-based multiplication. With the current setup, it is possible to deploy VMs for a single team without modifying the *SDL_exercise.yml*, but multiplication would allow more convenient team-based deployments. This could be achieved by developing the *SDL_exercise.yml* file to handle team based IPv4 network configurations and *SDL_environment.yml* to handle team-based environment-variables, such as, domain. Currently, a team-based CDX would require manual interaction after every team deployment for changing each team variables.

Also, in this proof of concept, highest hypervisor administrator access levels were used. In order to accommodate this toolset to work in bigger production environments, minimal required access levels should be defined when developing hypervisor specific deployment roles. Another feature that could be improved regarding user accounts, is the security of the credentials. In this proof of concept, the SDL files are populated with plain-text credentials for clarity and demonstration. As detailed, Ansible vault can be used to encrypt hypervisor credentials. However, once the content in conjunction with the described exercises start to grow, some proper credential management solution should be implemented.

As this proof of concept uses Ansible as a back-end, deployment is initiated by directly interacting with Ansible. Additional research could be done to find out, whether the interaction could also be handled by the SDL engine, thus eliminating the need to directly interact with the Ansible command line.

¹² <https://github.com/xjan76/SDL>

In regards to VM connectivity, this proof of concept utilized only IPv4. To further advance the SDL, IPv6 connection parameters could be implemented. This should remain as an optional feature, meaning that the exercise developer could decide, whether to use IPv6 or not. The implementation would require the addition of IPv6 parameter keys to *SDL_environment.yml*, *SDL_exercise.yml* and the respective changes should be made to the template files and network related tasks in hypervisor roles.

This proof of concept took a VM centric, low abstraction level approach in defining the core functionalities of the CDX and a higher abstraction level approach in describing the environment and deploying the exercise scenario. While the author maintains that this is enough to develop functional CDXs, some additional, higher abstraction level features, for example, storyline, which would enable a dynamic alteration of scenario components during the execution, could be implemented. Currently, similar functionality can be achieved, but on a lower abstraction level, meaning that the desired steps must be defined in VM configurations. As described in a paper [20] by the Norwegian University of Science and Technology, and displayed on Figure 2: Taxonomy of cyber ranges, there is a variety of high abstraction level artefacts, which could be implemented on top of the current solution. The currently developed SDL files could be expanded or additional SDL files could be developed to further enhance scenario development.

8 Conclusion

In this thesis, the author developed a proof of concept for a hypervisor agnostic Scenario Definition Language. To the authors knowledge, at the time of writing, there was no published literature on cyber defense exercise development frameworks, which prioritize exercise portability between cyber ranges, which differ in terms of hypervisor selection. Due to this, the author considers the proof of concept of this novel framework a valuable contribution in the field of developing cyber defense exercises.

Regarding the first research question, the author identified a set of conceptual requirements for the hypervisor agnostic SDL. In addition, as a result of a review on different data serialization languages, YAML was chosen as the primary data format for the framework.

Regarding the second research question, the author defined and developed a list of components, which enable the definition of the exercise environment and deployment information in a hypervisor agnostic manner. The first set of low abstraction level components provide the exercise developer means to develop exercise content in a hypervisor agnostic manner via dedicated Ansible directory. Those components include the ability to create both VM specific configurations and also universal, reusable functions, which can be used by different VMs. The second set of components, provide the exercise developer a higher abstraction level interface to define the general exercise environment variables and also define the deployable VM parameters. By using the combination of the two previously mentioned component sets, the author developed a small-scale example scenario. From the authors perspective, the designed framework provided all the necessary configuration options for scenario development in a hypervisor agnostic manner.

To answer the third research question, the author described the necessary steps required to develop and integrate new hypervisor platforms to the SDL. In this thesis, the author first detailed the hypervisor base requirements required for a successful SDL integration. The author then demonstrated the integration process of two different hypervisors: VMware

vSphere and KVM based OpenNebula; and validated the integrations by deploying the previously mentioned small-scale exercise to the said hypervisor platforms. This successful exercise deployment with the novel framework proves that cyber defense exercises can be developed in a hypervisor agnostic manner.

To reach the full operating functionality of the framework, additional research and development is definitely needed. While this thesis provides the core functionality for the framework, additional hypervisor integrations could be implemented and additional higher abstraction level artefacts could be developed. The author believes that those steps would further facilitate the introduction of the framework into different cyber ranges.

References

- [1] H. Taylor, "What is a cyber range?," 4 May 2021. [Online]. Available: <https://cybersecurityguide.org/resources/cyber-ranges/>. [Accessed 21 February 2022].
- [2] Red Hat, "What is KVM?," 21 March 2021. [Online]. Available: <https://www.redhat.com/en/topics/virtualization/what-is-KVM>. [Accessed 8 December 2021].
- [3] VMware, "VMware ESXi: The Purpose-Built Bare Metal Hypervisor," 2021. [Online]. Available: <https://www.vmware.com/products/esxi-and-esx.html>. [Accessed 8 December 2021].
- [4] Ansible, "Red Hat Ansible," 2021. [Online]. Available: <https://www.ansible.com>. [Accessed 8 December 2021].
- [5] HashiCorp, "Terraform," 2021. [Online]. Available: <https://www.terraform.io/>. [Accessed 8 December 2021].
- [6] HashiCorp, "Vagrant," 2021. [Online]. Available: <https://www.vagrantup.com/>. [Accessed 8 December 2021].
- [7] Progress, "Chef," [Online]. Available: <https://www.chef.io/products/chef-infra>. [Accessed 21 February 2022].
- [8] K. Sandeeb, "Terraform vs Ansible: Working, Difference, Provisioning," K21Academy, 10 July 2021. [Online]. Available: <https://k21academy.com/ansible/terraform-vs-ansible/>. [Accessed 8 December 2021].
- [9] VMware, "vSphere," [Online]. Available: <https://www.vmware.com/products/vsphere.html>. [Accessed 22 February 2022].
- [10] OpenNebula Systems, "Why OpenNebula?," [Online]. Available: <https://opennebula.io/discover/>. [Accessed 21 February 2022].
- [11] B. Posey, "Bare-metal hypervisor," March 2021. [Online]. Available: <https://searchservirtualization.techtarget.com/definition/bare-metal-hypervisor>. [Accessed 21 February 2022].

- [12] Amazon, "Cloud computing with AWS," 2021. [Online]. Available: <https://aws.amazon.com/what-is-aws>. [Accessed 8 December 2021].
- [13] Microsoft, "Microsoft Azure," 2021. [Online]. Available: <https://azure.microsoft.com/en-us/overview>. [Accessed 8 December 2021].
- [14] K. Puffers, T. Tuunanen, M. A. Rothenberger and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45-77, 8 December 2007.
- [15] Scopus, "Scopus," 2022. [Online]. Available: <https://www.scopus.com/home.uri>. [Accessed 07 April 2022].
- [16] E. Russo, G. Costa and A. Armando, "Building next generation Cyber Ranges with CRACK," *Computers & Security*, vol. 95, p. 101837, April 2020.
- [17] NIST, "Cyber Ranges," 13 February 2018. [Online]. Available: https://www.nist.gov/system/files/documents/2018/02/13/cyber_ranges.pdf. [Accessed 8 December 2021].
- [18] ECS, "Understanding Cyber Ranges: From Hype to Reality," March 2020. [Online]. Available: <https://ecs-org.eu/documents/publications/5fdb291cdf5e7.pdf>. [Accessed 8 December 2021].
- [19] Diateam, "What's a Cyber Range?," 2022. [Online]. Available: <https://www.diateam.net/what-is-a-cyber-range/>. [Accessed 07 April 2022].
- [20] B. Katt, M. Yamin and V. Gkioulus, "Cyber ranges and security testbeds: Scenarios, functions, tools and architecture," *Computers & Security*, vol. 88, p. 101636, January 2020.
- [21] T. Gustafsson and J. Almroth, "Cyber Range Automation Overview with a Case Study of CRATE," in *The 25th Nordic Conference on Secure IT Systems*, Online, 2020.
- [22] J. Vykopal, R. Ošlejšek, P. Celeda and M. Vizvary, "KYPO Cyber Range: Design and Use Cases," in *12th International Conference on Software Technologies*, Madrid, 2017.
- [23] A. Vallaots, "Federation of Cyber Ranges," 2017. [Online]. Available: https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=58426&year=2017. [Accessed 14 December 2021].
- [24] M. Leitner, M. Frank, W. Hotwagner, G. Lagner and O. Maurhart, "AIT Cyber Range: Flexible Cyber Security Environment for Exercises, Training and Research," in *Proceedings of the European Interdisciplinary Cybersecurity Conference*, Renners, 2020.
- [25] T. Cruz and P. Simones, "Down the Rabbit Hole: Fostering Active Learning through Guided Exploration of a SCADA Cyber Range," *Applied Sciences*, vol. 11, no. 20, p. 9509, 19 August 2021.

- [26] M. A. Haney, D. Conte de Leon, C. E. Goes and A. W. Krings, "ADLES: Specifying, deploying and sharing hands-on cyber-exercises," *Computers & Security*, vol. 74, no. 5, pp. 12-40, 2 January 2018.
- [27] B. Katt, M. M. Yamin and M. Nowostawski, "Serious games as a tool to model attack and defense scenarios for cyber-security exercises," *Computers & Security*, vol. 110, p. 102450, November 2021.
- [28] OASIS, "OASIS Topology and Orchestration Specification for Cloud Applications," 2020. [Online]. Available: <https://www.oasis-open.org/committees/tosca/faq.php>. [Accessed 14 December 2021].
- [29] Cloudify, "Connecting DevOps," 2021. [Online]. Available: <https://cloudify.co/product/>. [Accessed 14 December 2021].
- [30] OpenStack, "Heat Orchestration Template (HOT) Guide," 9 November 2020. [Online]. Available: https://docs.openstack.org/heat/rocky/template_guide/hot_guide.html. [Accessed 21 February 2022].
- [31] Puppet, "Puppet Enterprise," 2022. [Online]. Available: <https://puppet.com/products/puppet-enterprise/>. [Accessed 21 February 2022].
- [32] R. Beuran, C. Pham and D. Tang, "Cybersecurity Education and Training Support System: CyRIS," *IEICE Transactions on Information and Systems*, Vols. E101-D, no. 3, pp. 740-749, March 2018.
- [33] C. Pham, D. Tang, R. Beuran and K. Chinen, "CyRIS: A Cyber Range Instantiation System for Facilitating Security Training," in *The seventh international symposium on information and communication technology*, Ho Chi Minh, 2016.
- [34] I. Somarakis, M. Smyrlis, G. Spanoudakis and K. Fysarakis, "Model-Driven Cyber Range Training: A Cyber Security Assurance Perspective," in *1st Model-driven Simulation and Training Environments for Cybersecurity Workshop*, Luxembourg, 2019.
- [35] E. Trickel, F. Disperati, E. Gustafson and F. Kalantari, "Shell We Play A Game? CTF-as-a-service for Security Education," in *USENIX Workshop on Advances in Security Education*, Canada, 2017.
- [36] "Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events," 14 August 2017. [Online]. Available: [https://www.semanticscholar.org/paper/Security-Scenario-Generator-\(SecGen\)%3A-A-Framework-Schreuders-Shaw/b8ac02b9696a3bb560aae963bba0fa6813f44986](https://www.semanticscholar.org/paper/Security-Scenario-Generator-(SecGen)%3A-A-Framework-Schreuders-Shaw/b8ac02b9696a3bb560aae963bba0fa6813f44986). [Accessed 14 December 2021].

- [37] A. S. Raj, B. Alangot, S. Prabhu and K. Achuthan, "Scalable and lightweight CTF infrastructures using application containers," in *USENIX Advances in Security Education Workshop*, Austin, 2016.
- [38] D. Crockford, "Introducing JSON," [Online]. Available: <https://www.json.org/json-en.html>. [Accessed 22 Februar 2022].
- [39] GitHub, "YAML: YAML Ain't Markup Language™," [Online]. Available: <https://yaml.org/>. [Accessed 22 February 2022].
- [40] Mozilla, "XML introduction," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction. [Accessed 21 February 2022].
- [41] MessagePack, "MessagePack," [Online]. Available: <https://msgpack.org/index.html>. [Accessed 21 February 2022].
- [42] Google Developers, "Protocol Buffers," 2 February 2022. [Online]. Available: <https://developers.google.com/protocol-buffers/docs/overview>. [Accessed 21 February 2022].
- [43] Sharpened Productions, "Markup Language," 1 June 2011. [Online]. Available: https://techterms.com/definition/markup_language. [Accessed 21 February 2022].
- [44] Devopedia, "Data Serialization," 24 July 2020. [Online]. Available: <https://devopedia.org/data-serialization>. [Accessed 21 February 2022].
- [45] D. S. Nayanajith, "JSON vs YAML," 16 June 2020. [Online]. Available: <https://levelup.gitconnected.com/json-vs-yaml-6aa0243aefc6>. [Accessed 21 February 2022].
- [46] A. Gamela, "YAML vs. JSON: What is the difference?," 14 October 2021. [Online]. Available: <https://www.imaginarycloud.com/blog/yaml-vs-json-what-is-the-difference/>. [Accessed 21 February 2022].
- [47] M. Tournouj, "YAML: probably not so great after all," 15 April 2019. [Online]. Available: <https://www.arp242.net/yaml-config.html>. [Accessed 21 February 2022].
- [48] Ansible, "YAML Syntax," 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html. [Accessed 14 December 2021].
- [49] Ansible, "Ansible Integrations," 2022. [Online]. Available: <https://www.ansible.com/integrations?hsLang=en-us>. [Accessed 08 April 2022].
- [50] A. Rayome, "Ansible overtakes Chef and Puppet as the top cloud configuration management tool," 27 February 2019. [Online]. Available: <https://www.techrepublic.com/article/ansible-overtakes-chef-and-puppet-as-the-top-cloud-configuration-management-tool/>. [Accessed 08 April 2022].

- [51] CCDCOE, "Locked Shields," 2021. [Online]. Available: <https://ccdcoe.org/exercises/locked-shields/>. [Accessed 15 May 2021].
- [52] M. Bayer, "Mako Templates for Python," [Online]. Available: <https://www.makotemplates.org/>. [Accessed 15 April 2022].
- [53] K. Kaunis, "xjan76/SDL," 15 May 2022. [Online]. Available: <https://github.com/xjan76/SDL>. [Accessed 16 May 2022].
- [54] Ansible, "Encrypting content with Ansible Vault," 21 Decemeber 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/vault.html. [Accessed 17 April 2022].
- [55] Ansible, "Role Directory Structure," 30 April 2021. [Online]. Available: https://docs.ansible.com/ansible/2.8/user_guide/playbooks_reuse_roles.html#role-directory-structure. [Accessed 12 April 2022].
- [56] Ansible, "All modules," 11 October 2021. [Online]. Available: https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html. [Accessed 14 April 2022].
- [57] Ansible, "Desired State Configuration," 27 April 2022. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/windows_dsc.html. [Accessed 3 May 2022].
- [58] VMware, "VMware NSX-T Data Center Documentation," 2022. [Online]. Available: <https://docs.vmware.com/en/VMware-NSX-T-Data-Center/index.html>. [Accessed 17 April 2022].
- [59] Dell Inc., "Dell PowerStore Scalable All-Flash Storage," 2022. [Online]. Available: <https://www.dell.com/en-ee/dt/storage/powerstore-storage-appliance.htm#tab0=0>. [Accessed 17 April 2022].
- [60] VMware, "vSan," 2022. [Online]. Available: <https://www.vmware.com/products/vsan.html>. [Accessed 17 April 2022].
- [61] HashiCorp, "Build automated machine images," 2022. [Online]. Available: <https://www.packer.io/>. [Accessed 11 April 2022].
- [62] VMware, "Introduction to VMware Tools," 14 June 2021. [Online]. Available: <https://docs.vmware.com/en/VMware-Tools/11.3.0/com.vmware.vsphere.vmwaretools.doc/GUID-28C39A00-743B-4222-B697-6632E94A8E72.html>. [Accessed 11 April 2022].
- [63] Chocolatey Software, Inc., "THE PACKAGE MANAGER FOR WINDOWS," 2022. [Online]. Available: <https://chocolatey.org/>. [Accessed 11 April 2022].

- [64] Microsoft, "Get started with OpenSSH," 2022. [Online]. Available: https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse. [Accessed 11 April 2022].
- [65] HashiCorp, "Download Packer," 2022. [Online]. Available: <https://www.packer.io/downloads>. [Accessed 11 April 2022].
- [66] Microsoft, "Sysprep (Generalize) a Windows installation," 2022. [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/sysprep--generalize--a-windows-installation?view=windows-11>. [Accessed 3 May 2022].
- [67] GNU, "GNU Bash," 22 September 2020. [Online]. Available: <https://www.gnu.org/software/bash/>. [Accessed 24 April 2022].
- [68] Microsoft, "Install WSL," 6 April 2022. [Online]. Available: <https://docs.microsoft.com/en-us/windows/wsl/install>. [Accessed 24 April 2022].
- [69] Observium Limited, "Network monitoring with intuition," [Online]. Available: <https://www.observium.org/>. [Accessed 16 April 2022].
- [70] VMware, "Introduction to VMware Tools," 14 June 2021. [Online]. Available: <https://docs.vmware.com/en/VMware-Tools/12.0.0/com.vmware.vsphere.vmwaretools.doc/GUID-28C39A00-743B-4222-B697-6632E94A8E72.html>. [Accessed 12 May 2022].

Appendix 1: Ansible playbook for vSphere deployment

```
---
- hosts: localhost
  gather_facts: false

  pre_tasks:
    - set_fact:
        to_be_deployed: "{{ to_be_deployed|default([]) + [item] }}"
        with_inventory_hostnames: "{{ machine }}"

- hosts: "{{ hostvars['localhost'].to_be_deployed }}"
  serial: 1
  gather_facts: false

  roles:
    - core/populate_inventory

- hosts: deployable
  serial: 10
  gather_facts: false
  strategy: host_pinned

  module_defaults:
    vmware_guest: "{{ vmware_defaults }}"
    vmware_guest_disk: "{{ vmware_defaults }}"
    vmware_guest_tools_wait: "{{ vmware_defaults }}"
    vmware_guest_vm_shell: "{{ vmware_defaults }}"
    vmware_guest_network: "{{ vmware_defaults }}"
    vmware_guest_info: "{{ vmware_defaults }}"
    vmware_guest_powerstate: "{{ vmware_defaults }}"
    vmware_dvs_portgroup: "{{ vmware_defaults }}"

  roles:
    - core/vsphere
    - core/base_image
    - core/users
    - core/customization
```

Appendix 2: SDL_engine.py

```
from pathlib import Path
from mako.template import Template
import glob
import yaml

def read_yaml_from_file(file_path):
    with open(file_path, 'r') as stream:
        return yaml.safe_load(stream)

def write_variables(content, path, output_path=None):
    mytemplate = Template(filename=path)
    result = mytemplate.render(config=content)
    if output_path is None:
        output_path = path.removesuffix('.tmpl')
    output_file = open(output_path, "w")
    output_file.write(result)
    output_file.close()

def create_inventory(template_path, enviroment_path, exercise_path):
    files = glob.glob(template_path)
    content = {
        "enviroment": read_yaml_from_file(enviroment_path),
        "exercise": read_yaml_from_file(exercise_path)
    }
    for file in files:
        write_variables(content, file)

def save_group_variables(ansible_path, config_path):
    files = glob.glob(ansible_path)
    configuration = read_yaml_from_file(config_path)
    for file in files:
        write_variables(configuration, file)

def save_host_vars(ansible_path, config_path):
    templates_paths = glob.glob(ansible_path)
    templates = list(map(lambda x: x.split("/").pop().split(".")[0],
templates_paths))
```

```

content = read_yaml_from_file(config_path)
for virtual_machine in content.values():
    if virtual_machine['parent'] in templates:
        index = templates.index(virtual_machine['parent'])
        template_path = templates_paths[index]
        write_variables(virtual_machine, template_path,
Path(template_path).parent.joinpath(virtual_machine['hostname'] +
".yaml"))
    else:
        print("VM template unknown: " + virtual_machine['parent'])

save_group_variables("group_vars/*.tpl", 'sdl_environment.yaml')
save_host_vars("host_vars/*.tpl", 'sdl_exercise.yaml')
create_inventory("inventory.ini.tpl", 'sdl_environment.yaml',
'sdl_exercise.yaml')

```

Appendix 3: vSphere deployment example

```
kristjan@DESKTOP-992ALSL:/mnt/c/GIT/demo$ ansible-playbook vsphere.yml -e=deploy_mode=deploy
-e=machine=dc1
```

```
PLAY [localhost]
*****
*****
```

```
TASK [set_fact]
*****
*****
```

```
Friday 15 April 2022 14:16:30 +0300 (0:00:00.149) 0:00:00.149 *****
```

```
ok: [localhost] => (item=dc1)
```

```
PLAY [['dc1']]
*****
*****
```

```
TASK [core/populate_inventory : Add deployable group for single host]
*****
```

```
Friday 15 April 2022 14:16:30 +0300 (0:00:00.315) 0:00:00.465 *****
```

```
changed: [dc1]
```

```
PLAY [deployable]
*****
*****
```

```
TASK [setup]
*****
*****
```

```
Friday 15 April 2022 14:16:30 +0300 (0:00:00.082) 0:00:00.547 *****
```

```
ok: [dc1 -> localhost]
```

```
TASK [core/vsphere : include_tasks]
*****
```

```
Friday 15 April 2022 14:16:32 +0300 (0:00:01.504) 0:00:02.052 *****
```

```
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/vmware_guest.yml for dc1
```

```

TASK [core/vsphere : Define networks vars for vmware_guest]
*****
Friday 15 April 2022  14:16:32 +0300 (0:00:00.090)          0:00:02.142 *****
ok: [dc1]

TASK [core/vsphere : Start undeploy]
*****
Friday 15 April 2022  14:16:32 +0300 (0:00:00.072)          0:00:02.214 *****
skipping: [dc1]

TASK [core/vsphere : include_tasks]
*****
Friday 15 April 2022  14:16:32 +0300 (0:00:00.051)          0:00:02.266 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/portgroups.yml for dc1

TASK [core/vsphere : Making sure vSphere portgroups are present]
*****
Friday 15 April 2022  14:16:32 +0300 (0:00:00.117)          0:00:02.384 *****
ok: [dc1 -> localhost]

TASK [core/vsphere : include_tasks]
*****
Friday 15 April 2022  14:16:33 +0300 (0:00:01.206)          0:00:03.590 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/deploy.yml for dc1

TASK [core/vsphere : VM lookup]
*****
Friday 15 April 2022  14:16:34 +0300 (0:00:00.111)          0:00:03.701 *****
ok: [dc1 -> localhost]

TASK [core/vsphere : set_fact]
*****
Friday 15 April 2022  14:16:35 +0300 (0:00:01.111)          0:00:04.812 *****
ok: [dc1]

TASK [core/vsphere : Clone VM]
*****
Friday 15 April 2022  14:16:35 +0300 (0:00:00.083)          0:00:04.896 *****
changed: [dc1 -> localhost]

TASK [core/vsphere : Start VM]
*****

```

```

Friday 15 April 2022  14:22:18 +0300 (0:05:43.040)          0:05:47.937 *****
ok: [dcl -> localhost]

TASK [core/vsphere : Gather deployed machine info]
*****
Friday 15 April 2022  14:22:21 +0300 (0:00:02.922)          0:05:50.860 *****
ok: [dcl -> localhost]

TASK [core/vsphere : Errors occurred, redeploying]
*****
Friday 15 April 2022  14:22:22 +0300 (0:00:01.018)          0:05:51.879 *****
skipping: [dcl] => (item=undeploy.yml)
skipping: [dcl] => (item=deploy.yml)

TASK [core/vsphere : include_tasks]
*****
Friday 15 April 2022  14:22:22 +0300 (0:00:00.112)          0:05:51.991 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/network.yml for dcl

TASK [core/vsphere : Set vmware tools connection]
*****
Friday 15 April 2022  14:22:22 +0300 (0:00:00.097)          0:05:52.089 *****
ok: [dcl]

TASK [core/vsphere : Waiting for connection]
*****
Friday 15 April 2022  14:22:22 +0300 (0:00:00.067)          0:05:52.156 *****
[WARNING]: Reset is not implemented for this connection
ok: [dcl]

TASK [core/vsphere : Gathering facts]
*****
Friday 15 April 2022  14:22:49 +0300 (0:00:27.076)          0:06:19.233 *****
ok: [dcl]

TASK [core/vsphere : include_tasks]
*****
Friday 15 April 2022  14:23:29 +0300 (0:00:40.073)          0:06:59.306 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/configuration/windows_cli.yml for dcl

TASK [core/vsphere : Configure interfaces]
*****

```

```

Friday 15 April 2022  14:23:29 +0300 (0:00:00.180)          0:06:59.486 *****
changed: [dcl] => (item={'name': 'vlan68', 'ipv4': '10.80.68.31/24', 'ipv4_gateway':
'10.80.68.1', 'vlan': 68})

TASK [core/vsphere : include_tasks]
*****
Friday 15 April 2022  14:23:49 +0300 (0:00:19.277)          0:07:18.763 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/connection.yml for dcl

TASK [core/vsphere : Setting connection IP]
*****
Friday 15 April 2022  14:23:49 +0300 (0:00:00.115)          0:07:18.879 *****
ok: [dcl]

TASK [core/vsphere : Setting Ansible connection]
*****
Friday 15 April 2022  14:23:49 +0300 (0:00:00.060)          0:07:18.940 *****
ok: [dcl]

TASK [core/vsphere : Waiting for connection]
*****
Friday 15 April 2022  14:23:49 +0300 (0:00:00.057)          0:07:18.998 *****
ok: [dcl]

TASK [core/vsphere : set_fact]
*****
Friday 15 April 2022  14:23:55 +0300 (0:00:06.179)          0:07:25.177 *****
ok: [dcl]

TASK [core/vsphere : Gathering facts...]
*****
Friday 15 April 2022  14:23:55 +0300 (0:00:00.057)          0:07:25.235 *****
ok: [dcl]

TASK [core/base_image : Configure Windows machine]
*****
Friday 15 April 2022  14:24:06 +0300 (0:00:11.258)          0:07:36.493 *****
included: /mnt/c/GIT/demo/roles/core/base_image/tasks/windows.yml for dcl

TASK [core/base_image : Set correct time]
*****
Friday 15 April 2022  14:24:06 +0300 (0:00:00.119)          0:07:36.613 *****

```


changed: [dc1]

TASK [core/base_image : Apply windows baseconf]

Friday 15 April 2022 14:24:25 +0300 (0:00:18.240) 0:07:54.854 *****

changed: [dc1]

TASK [core/base_image : Enable RDP service]

Friday 15 April 2022 14:24:36 +0300 (0:00:11.649) 0:08:06.503 *****

changed: [dc1]

TASK [core/base_image : Enable RDP firewall]

Friday 15 April 2022 14:24:44 +0300 (0:00:07.899) 0:08:14.403 *****

changed: [dc1]

TASK [core/base_image : Changing Windows hostname]

Friday 15 April 2022 14:24:54 +0300 (0:00:10.200) 0:08:24.603 *****

ok: [dc1]

TASK [core/base_image : Rebooting, if required]

Friday 15 April 2022 14:25:00 +0300 (0:00:05.723) 0:08:30.327 *****

skipping: [dc1]

TASK [core/base_image : Configure Linux machine]

Friday 15 April 2022 14:25:00 +0300 (0:00:00.078) 0:08:30.405 *****

skipping: [dc1]

TASK [core/users : Change Windows users' passwords]

Friday 15 April 2022 14:25:00 +0300 (0:00:00.055) 0:08:30.461 *****

included: /mnt/c/GIT/demo/roles/core/users/tasks/windows.yml for dc1

TASK [core/users : Verifying regular accounts]

Friday 15 April 2022 14:25:00 +0300 (0:00:00.103) 0:08:30.564 *****

changed: [dc1] => (item={'username': 'Administrator', 'password': 'Password.123'})

```

TASK [core/users : Verifying DC accounts]
*****
Friday 15 April 2022  14:25:09 +0300 (0:00:08.563)          0:08:39.127 *****
skipping: [dcl] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [core/users : Creating a profile for required accounts]
*****
Friday 15 April 2022  14:25:09 +0300 (0:00:00.071)          0:08:39.199 *****
ok: [dcl] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [Start VM customization]
*****
Friday 15 April 2022  14:25:15 +0300 (0:00:06.248)          0:08:45.447 *****

TASK [conf/install_primary_domain_controller : Install AD-Domain-Services role]
*****
Friday 15 April 2022  14:25:16 +0300 (0:00:00.248)          0:08:45.696 *****
changed: [dcl]

TASK [conf/install_primary_domain_controller : Creating new Windows domain]
*****
Friday 15 April 2022  14:26:17 +0300 (0:01:01.703)          0:09:47.399 *****
changed: [dcl]

TASK [conf/install_primary_domain_controller : Rebooting, if required]
*****
Friday 15 April 2022  14:27:14 +0300 (0:00:57.132)          0:10:44.532 *****
changed: [dcl]

TASK [conf/install_primary_domain_controller : Checking PDC availability]
*****
Friday 15 April 2022  14:28:34 +0300 (0:01:19.280)          0:12:03.813 *****
changed: [dcl]

TASK [conf/install_primary_domain_controller : Gathering facts]
*****
Friday 15 April 2022  14:36:44 +0300 (0:08:09.953)          0:20:13.767 *****
ok: [dcl]

TASK [Checking DC status]
*****
Friday 15 April 2022  14:37:18 +0300 (0:00:34.597)          0:20:48.364 *****

```

TASK [conf/dc_check : Waiting for Active Directory services]

Friday 15 April 2022 14:37:18 +0300 (0:00:00.144) 0:20:48.509 *****

changed: [dc1]

TASK [conf/dc_check : Getting domain info]

Friday 15 April 2022 14:37:26 +0300 (0:00:07.887) 0:20:56.397 *****

changed: [dc1]

TASK [vm/dc1 : Adding required Users, groups and OUs for Observium]

Friday 15 April 2022 14:37:36 +0300 (0:00:09.473) 0:21:05.870 *****

changed: [dc1]

PLAY RECAP

dc1 : ok=41 changed=15 unreachable=0 failed=0 skipped=5
rescued=0 ignored=0

localhost : ok=1 changed=0 unreachable=0 failed=0 skipped=0
rescued=0 ignored=0

Friday 15 April 2022 14:37:44 +0300 (0:00:08.722) 0:21:14.593 *****

Appendix 4: OpenNebula deployment example

```
kristjan@DESKTOP-992ALSL:/mnt/c/GIT/demo$ ansible-playbook opennebula.yml -  
e=deploy_mode=deploy -e=machine=dc1
```

```
PLAY [localhost]  
*****  
*****
```

```
TASK [set_fact]  
*****  
*****
```

```
Saturday 16 April 2022 13:47:18 +0300 (0:00:00.141) 0:00:00.141 *****
```

```
ok: [localhost] => (item=dc1)
```

```
PLAY [['dc1']]  
*****  
*****
```

```
TASK [core/populate_inventory : Add deployable group for single host]  
*****
```

```
Saturday 16 April 2022 13:47:18 +0300 (0:00:00.314) 0:00:00.456 *****
```

```
changed: [dc1]
```

```
PLAY [deployable]  
*****  
*****
```

```
TASK [setup]  
*****  
*****
```

```
Saturday 16 April 2022 13:47:18 +0300 (0:00:00.084) 0:00:00.540 *****
```

```
ok: [dc1 -> localhost]
```

```
TASK [core/opennebula : Start undeploy]  
*****
```

```
Saturday 16 April 2022 13:47:20 +0300 (0:00:01.422) 0:00:01.963 *****
```

```
skipping: [dc1]
```

```

TASK [core/opennebula : include_tasks]
*****
Saturday 16 April 2022  13:47:20 +0300 (0:00:00.047)      0:00:02.011 *****
included: /mnt/c/GIT/demo/roles/core/opennebula/tasks/deploy.yml for dc1

TASK [core/opennebula : Deployig VM]
*****
Saturday 16 April 2022  13:47:20 +0300 (0:00:00.136)      0:00:02.147 *****
changed: [dc1 -> localhost]

TASK [core/opennebula : Creating a new template for Exchange server]
*****
Saturday 16 April 2022  13:51:42 +0300 (0:04:22.464)      0:04:24.612 *****
skipping: [dc1]

TASK [core/opennebula : Deployig Exchange VM]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.064)      0:04:24.677 *****
skipping: [dc1]

TASK [core/opennebula : set_fact]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.050)      0:04:24.727 *****
ok: [dc1]

TASK [core/opennebula : Print VM properties]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.072)      0:04:24.800 *****
ok: [dc1 -> localhost] => changed=false

msg: result

TASK [core/opennebula : include_tasks]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.056)      0:04:24.856 *****
included: /mnt/c/GIT/demo/roles/core/opennebula/tasks/connection.yml for dc1

TASK [core/opennebula : Setting initial connection]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.113)      0:04:24.969 *****
ok: [dc1]

```

```

TASK [core/opennebula : Set networked Ansible connection]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.059)          0:04:25.029 *****
ok: [dc1]

TASK [core/opennebula : Waiting for system to become reachable]
*****
Saturday 16 April 2022  13:51:43 +0300 (0:00:00.059)          0:04:25.089 *****
ok: [dc1]

TASK [core/opennebula : Gathering facts...]
*****
Saturday 16 April 2022  13:55:52 +0300 (0:04:09.199)          0:08:34.289 *****
ok: [dc1]

TASK [core/opennebula : include_tasks]
*****
Saturday 16 April 2022  13:56:10 +0300 (0:00:17.723)          0:08:52.013 *****
included: /mnt/c/GIT/demo/roles/core/opennebula/tasks/network.yml for dc1

TASK [core/opennebula : Waiting for system to become reachable]
*****
Saturday 16 April 2022  13:56:10 +0300 (0:00:00.103)          0:08:52.116 *****
ok: [dc1]

TASK [core/opennebula : include_tasks]
*****
Saturday 16 April 2022  13:56:13 +0300 (0:00:03.056)          0:08:55.173 *****
included: /mnt/c/GIT/demo/roles/core/opennebula/tasks/configuration/windows_cli.yml for dc1

TASK [core/opennebula : Configure NIC-s]
*****
Saturday 16 April 2022  13:56:13 +0300 (0:00:00.214)          0:08:55.388 *****
fatal: [dc1]: UNREACHABLE! => changed=false

  msg: |-
    Data could not be sent to remote host. Make sure this host can be reached over ssh: #<
    CLIXML

  skip_reason: Host dc1 is unreachable

  unreachable: true

TASK [core/opennebula : include_tasks]
*****

```

Saturday 16 April 2022 13:56:54 +0300 (0:00:41.066) 0:09:36.455 *****
included: /mnt/c/GIT/demo/roles/core/opennebula/tasks/new_connection.yml for dc1

TASK [core/opennebula : Setting new connection IP]

Saturday 16 April 2022 13:56:54 +0300 (0:00:00.112) 0:09:36.567 *****
ok: [dc1]

TASK [core/opennebula : Set networked Ansible connection]

Saturday 16 April 2022 13:56:54 +0300 (0:00:00.055) 0:09:36.622 *****
ok: [dc1]

TASK [core/opennebula : Waiting for system to become reachable]

Saturday 16 April 2022 13:56:55 +0300 (0:00:00.046) 0:09:36.669 *****
ok: [dc1]

TASK [core/opennebula : set_fact]

Saturday 16 April 2022 13:56:57 +0300 (0:00:02.392) 0:09:39.061 *****
ok: [dc1]

TASK [core/opennebula : Gathering facts...]

Saturday 16 April 2022 13:56:57 +0300 (0:00:00.055) 0:09:39.117 *****
ok: [dc1]

TASK [core/base_image : Configure Windows machine]

Saturday 16 April 2022 13:57:02 +0300 (0:00:04.928) 0:09:44.045 *****
included: /mnt/c/GIT/demo/roles/core/base_image/tasks/windows.yml for dc1

TASK [core/base_image : Set correct time]

Saturday 16 April 2022 13:57:02 +0300 (0:00:00.123) 0:09:44.169 *****
changed: [dc1]

TASK [core/base_image : Apply windows baseconf]

Saturday 16 April 2022 13:57:16 +0300 (0:00:13.650) 0:09:57.819 *****
changed: [dc1]

```

TASK [core/base_image : Enable RDP service]
*****
Saturday 16 April 2022  13:57:23 +0300 (0:00:07.489)          0:10:05.309 *****
changed: [dc1]

TASK [core/base_image : Enable RDP firewall]
*****
Saturday 16 April 2022  13:57:26 +0300 (0:00:02.612)          0:10:07.921 *****
changed: [dc1]

TASK [core/base_image : Changing Windows hostname]
*****
Saturday 16 April 2022  13:57:31 +0300 (0:00:04.804)          0:10:12.725 *****
changed: [dc1]

TASK [core/base_image : Rebooting, if required]
*****
Saturday 16 April 2022  13:57:33 +0300 (0:00:02.178)          0:10:14.903 *****
changed: [dc1]

TASK [core/base_image : Configure Linux machine]
*****
Saturday 16 April 2022  13:58:22 +0300 (0:00:49.339)          0:11:04.243 *****
skipping: [dc1]

TASK [core/users : Change Windows users' passwords]
*****
Saturday 16 April 2022  13:58:22 +0300 (0:00:00.041)          0:11:04.284 *****
included: /mnt/c/GIT/demo/roles/core/users/tasks/windows.yml for dc1

TASK [core/users : Verifying regular accounts]
*****
Saturday 16 April 2022  13:58:22 +0300 (0:00:00.103)          0:11:04.387 *****
changed: [dc1] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [core/users : Verifying DC accounts]
*****
Saturday 16 April 2022  13:58:36 +0300 (0:00:13.496)          0:11:17.884 *****
skipping: [dc1] => (item={'username': 'Administrator', 'password': 'Password.123'})

```



```

TASK [core/users : Creating a profile for required accounts]
*****
Saturday 16 April 2022  13:58:36 +0300 (0:00:00.068)          0:11:17.952 *****
ok: [dc1] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [Start VM customization]
*****
Saturday 16 April 2022  13:58:39 +0300 (0:00:02.980)          0:11:20.933 *****

TASK [conf/install_primary_domain_controller : Install AD-Domain-Services role]
*****
Saturday 16 April 2022  13:58:39 +0300 (0:00:00.251)          0:11:21.184 *****
changed: [dc1]

TASK [conf/install_primary_domain_controller : Creating new Windows domain]
*****
Saturday 16 April 2022  14:00:28 +0300 (0:01:49.262)          0:13:10.447 *****
changed: [dc1]

TASK [conf/install_primary_domain_controller : Rebooting, if required]
*****
Saturday 16 April 2022  14:01:32 +0300 (0:01:03.588)          0:14:14.036 *****
changed: [dc1]

TASK [conf/install_primary_domain_controller : Checking PDC availability]
*****
Saturday 16 April 2022  14:07:43 +0300 (0:06:11.536)          0:20:25.572 *****
changed: [dc1]

TASK [conf/install_primary_domain_controller : Gathering facts]
*****
Saturday 16 April 2022  14:10:51 +0300 (0:03:07.847)          0:23:33.419 *****
ok: [dc1]

TASK [Checking DC status]
*****
**
Saturday 16 April 2022  14:11:25 +0300 (0:00:33.832)          0:24:07.252 *****

TASK [conf/dc_check : Waiting for Active Directory services]
*****
Saturday 16 April 2022  14:11:25 +0300 (0:00:00.149)          0:24:07.401 *****
changed: [dc1]

```

TASK [conf/dc_check : Getting domain info]

Saturday 16 April 2022 14:11:28 +0300 (0:00:02.785) 0:24:10.187 *****

changed: [dc1]

TASK [vm/dc1 : Adding required Users, groups and OUs for Observium]

Saturday 16 April 2022 14:11:33 +0300 (0:00:05.404) 0:24:15.591 *****

changed: [dc1]

PLAY RECAP

dc1		: ok=38	changed=16	unreachable=1	failed=0	skipped=6
rescued=0	ignored=0					
localhost		: ok=1	changed=0	unreachable=0	failed=0	skipped=0
rescued=0	ignored=0					

Saturday 16 April 2022 14:11:37 +0300 (0:00:03.818) 0:24:19.410 *****

Appendix 5: vSphere dual deployment example

```
kristjan@DESKTOP-992ALSL:/mnt/c/GIT/demo$ ansible-playbook vsphere.yml -e=deploy_mode=redeploy -e=machine=ws1,ws2
```

```
PLAY [localhost]
*****
*****
```

```
TASK [set_fact]
*****
*****
```

```
Wednesday 11 May 2022 23:44:16 +0300 (0:00:00.154) 0:00:00.154 *****
```

```
ok: [localhost] => (item=ws1)
```

```
ok: [localhost] => (item=ws2)
```

```
PLAY [['ws1', 'ws2']]
*****
*****
```

```
TASK [core/populate_inventory : Add deployable group for single host]
*****
```

```
Wednesday 11 May 2022 23:44:16 +0300 (0:00:00.363) 0:00:00.518 *****
```

```
changed: [ws1]
```

```
PLAY [['ws1', 'ws2']]
*****
*****
```

```
TASK [core/populate_inventory : Add deployable group for single host]
*****
```

```
Wednesday 11 May 2022 23:44:16 +0300 (0:00:00.070) 0:00:00.588 *****
```

```
changed: [ws2]
```

```
PLAY [deployable]
*****
*****
```

```
TASK [setup]
*****
*****
```

```
Wednesday 11 May 2022 23:44:16 +0300 (0:00:00.085) 0:00:00.674 *****
```

```
TASK [setup]
*****
*****
```

```
Wednesday 11 May 2022 23:44:16 +0300 (0:00:00.046) 0:00:00.720 *****
```

```
ok: [ws2 -> localhost]
```

```
TASK [core/vsphere : include_tasks]
*****
```

```
Wednesday 11 May 2022 23:44:18 +0300 (0:00:01.298) 0:00:02.019 *****
```

```
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/vmware_guest.yml for ws2
```

```
TASK [core/vsphere : Define networks vars for vmware_guest]
*****
```

```
Wednesday 11 May 2022 23:44:18 +0300 (0:00:00.110) 0:00:02.129 *****
```

```
ok: [ws2]
```

```
TASK [core/vsphere : Start undeploy]
*****
```

```
Wednesday 11 May 2022 23:44:18 +0300 (0:00:00.082) 0:00:02.211 *****
```

```
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/undeploy.yml for ws2
```

```
TASK [core/vsphere : Undeploy VM]
*****
```

```
Wednesday 11 May 2022 23:44:18 +0300 (0:00:00.142) 0:00:02.354 *****
```

```
TASK [core/vsphere : Undeploy VM]
*****
```

```
ok: [ws1 -> localhost]
```

```
TASK [core/vsphere : include_tasks]
*****
```

```
Wednesday 11 May 2022 23:44:19 +0300 (0:00:00.931) 0:00:03.285 *****
```

```
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/vmware_guest.yml for ws1
```

```
TASK [core/vsphere : Define networks vars for vmware_guest]
*****
```

```
Wednesday 11 May 2022 23:44:19 +0300 (0:00:00.112) 0:00:03.398 *****
```

```
ok: [ws1]
```

TASK [core/vsphere : Start undeploy]

Wednesday 11 May 2022 23:44:19 +0300 (0:00:00.076) 0:00:03.474 *****

TASK [core/vsphere : Start undeploy]

ok: [ws2 -> localhost]

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/undeploy.yml for ws1

TASK [core/vsphere : Undeploy VM]

Wednesday 11 May 2022 23:44:19 +0300 (0:00:00.189) 0:00:03.664 *****

TASK [core/vsphere : include_tasks]

Wednesday 11 May 2022 23:44:19 +0300 (0:00:00.040) 0:00:03.705 *****

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/portgroups.yml for ws2

TASK [core/vsphere : Making sure vSphere portgroups are present]

Wednesday 11 May 2022 23:44:20 +0300 (0:00:00.128) 0:00:03.833 *****

TASK [core/vsphere : Making sure vSphere portgroups are present]

ok: [ws1 -> localhost]

TASK [core/vsphere : include_tasks]

Wednesday 11 May 2022 23:44:20 +0300 (0:00:00.882) 0:00:04.716 *****

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/portgroups.yml for ws1

TASK [core/vsphere : Making sure vSphere portgroups are present]

Wednesday 11 May 2022 23:44:21 +0300 (0:00:00.121) 0:00:04.837 *****

TASK [core/vsphere : Making sure vSphere portgroups are present]

ok: [ws2 -> localhost]

TASK [core/vsphere : include_tasks]

Wednesday 11 May 2022 23:44:21 +0300 (0:00:00.194) 0:00:05.032 *****

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/deploy.yml for ws2

TASK [core/vsphere : VM lookup]

Wednesday 11 May 2022 23:44:21 +0300 (0:00:00.111) 0:00:05.143 *****

TASK [core/vsphere : VM lookup]

ok: [ws1 -> localhost]

TASK [core/vsphere : include_tasks]

Wednesday 11 May 2022 23:44:22 +0300 (0:00:00.786) 0:00:05.930 *****

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/deploy.yml for ws1

TASK [core/vsphere : VM lookup]

Wednesday 11 May 2022 23:44:22 +0300 (0:00:00.128) 0:00:06.059 *****

TASK [core/vsphere : VM lookup]

ok: [ws2 -> localhost]

TASK [core/vsphere : set_fact]

Wednesday 11 May 2022 23:44:22 +0300 (0:00:00.226) 0:00:06.285 *****

ok: [ws2]

TASK [core/vsphere : Clone VM]

Wednesday 11 May 2022 23:44:22 +0300 (0:00:00.075) 0:00:06.361 *****

TASK [core/vsphere : Clone VM]

ok: [ws1 -> localhost]

TASK [core/vsphere : set_fact]

Wednesday 11 May 2022 23:44:23 +0300 (0:00:00.700) 0:00:07.062 *****

ok: [ws1]

TASK [core/vsphere : Clone VM]

Wednesday 11 May 2022 23:44:23 +0300 (0:00:00.072) 0:00:07.134 *****

TASK [core/vsphere : Clone VM]

changed: [ws2 -> localhost]

TASK [core/vsphere : Start VM]

Wednesday 11 May 2022 23:49:54 +0300 (0:05:30.742) 0:05:37.877 *****

ok: [ws2 -> localhost]

TASK [core/vsphere : Gather deployed machine info]

Wednesday 11 May 2022 23:49:57 +0300 (0:00:02.973) 0:05:40.851 *****

ok: [ws2 -> localhost]

TASK [core/vsphere : Errors occurred, redeploying]

Wednesday 11 May 2022 23:49:57 +0300 (0:00:00.879) 0:05:41.730 *****

skipping: [ws2] => (item=undeploy.yml)

skipping: [ws2] => (item=deploy.yml)

TASK [core/vsphere : include_tasks]

Wednesday 11 May 2022 23:49:58 +0300 (0:00:00.112) 0:05:41.843 *****

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/network.yml for ws2

TASK [core/vsphere : Set vmware tools connection]

Wednesday 11 May 2022 23:49:58 +0300 (0:00:00.105) 0:05:41.949 *****

ok: [ws2]

TASK [core/vsphere : Waiting for connection]

Wednesday 11 May 2022 23:49:58 +0300 (0:00:00.075) 0:05:42.024 *****

[WARNING]: Reset is not implemented for this connection

ok: [ws2]

TASK [core/vsphere : Gathering facts]

Wednesday 11 May 2022 23:50:21 +0300 (0:00:23.625) 0:06:05.649 *****

ok: [ws2]

```
TASK [core/vsphere : include_tasks]
*****
Wednesday 11 May 2022  23:50:55 +0300 (0:00:33.821)          0:06:39.470 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/configuration/windows_cli.yml for ws2
```

```
TASK [core/vsphere : Configure interfaces]
*****
Wednesday 11 May 2022  23:50:55 +0300 (0:00:00.150)          0:06:39.621 *****
changed: [ws2] => (item={'name': 'vlan68', 'ipv4': '10.80.68.36/24', 'ipv4_gateway': '10.80.68.1', 'vlan': 68})
```

```
TASK [core/vsphere : include_tasks]
*****
Wednesday 11 May 2022  23:51:17 +0300 (0:00:21.410)          0:07:01.031 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/connection.yml for ws2
```

```
TASK [core/vsphere : Setting connection IP]
*****
Wednesday 11 May 2022  23:51:17 +0300 (0:00:00.094)          0:07:01.126 *****
ok: [ws2]
```

```
TASK [core/vsphere : Setting Ansible connection]
*****
Wednesday 11 May 2022  23:51:17 +0300 (0:00:00.058)          0:07:01.185 *****
ok: [ws2]
```

```
TASK [core/vsphere : Waiting for connection]
*****
Wednesday 11 May 2022  23:51:17 +0300 (0:00:00.054)          0:07:01.239 *****
ok: [ws2]
```

```
TASK [core/vsphere : set_fact]
*****
Wednesday 11 May 2022  23:51:23 +0300 (0:00:06.355)          0:07:07.595 *****
ok: [ws2]
```

```
TASK [core/vsphere : Gathering facts...]
*****
Wednesday 11 May 2022  23:51:23 +0300 (0:00:00.052)          0:07:07.648 *****
ok: [ws2]
```



```
TASK [core/base_image : Configure Windows machine]
*****
Wednesday 11 May 2022  23:51:38 +0300 (0:00:14.981)      0:07:22.629 *****
included: /mnt/c/GIT/demo/roles/core/base_image/tasks/windows.yml for ws2
```

```
TASK [core/base_image : Set correct time]
*****
Wednesday 11 May 2022  23:51:38 +0300 (0:00:00.114)      0:07:22.743 *****
changed: [ws2]
```

```
TASK [core/base_image : Apply windows baseconf]
*****
Wednesday 11 May 2022  23:52:11 +0300 (0:00:32.891)      0:07:55.634 *****
```

```
TASK [core/base_image : Apply windows baseconf]
*****
changed: [ws1 -> localhost]
```

```
TASK [core/vsphere : Start VM]
*****
Wednesday 11 May 2022  23:52:36 +0300 (0:00:25.126)      0:08:20.761 *****
ok: [ws1 -> localhost]
```

```
TASK [core/vsphere : Gather deployed machine info]
*****
Wednesday 11 May 2022  23:52:39 +0300 (0:00:02.805)      0:08:23.567 *****
ok: [ws1 -> localhost]
```

```
TASK [core/vsphere : Errors occurred, redeploying]
*****
Wednesday 11 May 2022  23:52:40 +0300 (0:00:01.142)      0:08:24.709 *****
skipping: [ws1] => (item=undeploy.yml)
skipping: [ws1] => (item=deploy.yml)
```

```
TASK [core/vsphere : include_tasks]
*****
Wednesday 11 May 2022  23:52:41 +0300 (0:00:00.111)      0:08:24.820 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/network.yml for ws1
```

```
TASK [core/vsphere : Set vmware tools connection]
*****
Wednesday 11 May 2022  23:52:41 +0300 (0:00:00.094)      0:08:24.915 *****
```

ok: [wsl]

TASK [core/vsphere : Waiting for connection]

Wednesday 11 May 2022 23:52:41 +0300 (0:00:00.066) 0:08:24.982 *****

[WARNING]: Reset is not implemented for this connection

TASK [core/vsphere : Waiting for connection]

changed: [ws2]

TASK [core/base_image : Enable RDP service]

Wednesday 11 May 2022 23:52:57 +0300 (0:00:15.948) 0:08:40.930 *****

TASK [core/base_image : Enable RDP service]

ok: [wsl]

TASK [core/vsphere : Gathering facts]

Wednesday 11 May 2022 23:53:00 +0300 (0:00:03.737) 0:08:44.668 *****

TASK [core/vsphere : Gathering facts]

changed: [ws2]

TASK [core/base_image : Enable RDP firewall]

Wednesday 11 May 2022 23:53:04 +0300 (0:00:03.918) 0:08:48.587 *****

changed: [ws2]

TASK [core/base_image : Changing Windows hostname]

Wednesday 11 May 2022 23:53:14 +0300 (0:00:10.108) 0:08:58.696 *****

ok: [ws2]

TASK [core/base_image : Rebooting, if required]

Wednesday 11 May 2022 23:53:20 +0300 (0:00:05.466) 0:09:04.163 *****

skipping: [ws2]

```

TASK [core/base_image : Configure Linux machine]
*****
Wednesday 11 May 2022  23:53:20 +0300 (0:00:00.061)      0:09:04.224 *****
skipping: [ws2]

TASK [core/users : Change Windows users' passwords]
*****
Wednesday 11 May 2022  23:53:20 +0300 (0:00:00.048)      0:09:04.273 *****
included: /mnt/c/GIT/demo/roles/core/users/tasks/windows.yml for ws2

TASK [core/users : Verifying regular accounts]
*****
Wednesday 11 May 2022  23:53:20 +0300 (0:00:00.097)      0:09:04.370 *****

TASK [core/users : Verifying regular accounts]
*****
ok: [ws1]

TASK [core/vsphere : include_tasks]
*****
Wednesday 11 May 2022  23:53:27 +0300 (0:00:07.282)      0:09:11.652 *****
included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/configuration/windows_cli.yml for ws1

TASK [core/vsphere : Configure interfaces]
*****
Wednesday 11 May 2022  23:53:28 +0300 (0:00:00.180)      0:09:11.833 *****

TASK [core/vsphere : Configure interfaces]
*****
ok: [ws2] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [core/users : Verifying DC accounts]
*****
Wednesday 11 May 2022  23:53:28 +0300 (0:00:00.759)      0:09:12.593 *****
skipping: [ws2] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [core/users : Creating a profile for required accounts]
*****
Wednesday 11 May 2022  23:53:28 +0300 (0:00:00.064)      0:09:12.658 *****
ok: [ws2] => (item={'username': 'Administrator', 'password': 'Password.123'})

```

TASK [Start VM customization]

*

Wednesday 11 May 2022 23:53:34 +0300 (0:00:06.056) 0:09:18.714 *****

TASK [conf/join_domain : Adding Windows machines to domain]

Wednesday 11 May 2022 23:53:35 +0300 (0:00:00.235) 0:09:18.950 *****

included: /mnt/c/GIT/demo/roles/conf/join_domain/tasks/windows.yml for ws2

TASK [conf/join_domain : Adding machine to domain]

Wednesday 11 May 2022 23:53:35 +0300 (0:00:00.109) 0:09:19.060 *****

changed: [ws2]

TASK [conf/join_domain : Rebooting machine]

Wednesday 11 May 2022 23:53:42 +0300 (0:00:07.411) 0:09:26.471 *****

TASK [conf/join_domain : Rebooting machine]

changed: [ws1] => (item={'name': 'vlan68', 'ipv4': '10.80.68.35/24', 'ipv4_gateway': '10.80.68.1', 'vlan': 68})

TASK [core/vsphere : include_tasks]

Wednesday 11 May 2022 23:53:43 +0300 (0:00:00.459) 0:09:26.931 *****

included: /mnt/c/GIT/demo/roles/core/vsphere/tasks/connection.yml for ws1

TASK [core/vsphere : Setting connection IP]

Wednesday 11 May 2022 23:53:43 +0300 (0:00:00.095) 0:09:27.026 *****

ok: [ws1]

TASK [core/vsphere : Setting Ansible connection]

Wednesday 11 May 2022 23:53:43 +0300 (0:00:00.058) 0:09:27.085 *****

ok: [ws1]

TASK [core/vsphere : Waiting for connection]

Wednesday 11 May 2022 23:53:43 +0300 (0:00:00.058) 0:09:27.144 *****

ok: [ws1]

```

TASK [core/vsphere : set_fact]
*****
Wednesday 11 May 2022  23:53:46 +0300 (0:00:02.921)          0:09:30.065 *****
ok: [wsl]

TASK [core/vsphere : Gathering facts...]
*****
Wednesday 11 May 2022  23:53:46 +0300 (0:00:00.065)          0:09:30.131 *****
ok: [wsl]

TASK [core/base_image : Configure Windows machine]
*****
Wednesday 11 May 2022  23:53:57 +0300 (0:00:11.202)          0:09:41.333 *****
included: /mnt/c/GIT/demo/roles/core/base_image/tasks/windows.yml for wsl

TASK [core/base_image : Set correct time]
*****
Wednesday 11 May 2022  23:53:57 +0300 (0:00:00.094)          0:09:41.428 *****
changed: [wsl]

TASK [core/base_image : Apply windows baseconf]
*****
Wednesday 11 May 2022  23:54:11 +0300 (0:00:14.307)          0:09:55.735 *****
changed: [wsl]

TASK [core/base_image : Enable RDP service]
*****
Wednesday 11 May 2022  23:54:24 +0300 (0:00:12.312)          0:10:08.048 *****
changed: [wsl]

TASK [core/base_image : Enable RDP firewall]
*****
Wednesday 11 May 2022  23:54:26 +0300 (0:00:02.652)          0:10:10.700 *****
changed: [wsl]

TASK [core/base_image : Changing Windows hostname]
*****
Wednesday 11 May 2022  23:54:32 +0300 (0:00:05.403)          0:10:16.103 *****
ok: [wsl]

```

```

TASK [core/base_image : Rebooting, if required]
*****
Wednesday 11 May 2022  23:54:34 +0300 (0:00:01.915)          0:10:18.019 *****
skipping: [wsl]

TASK [core/base_image : Configure Linux machine]
*****
Wednesday 11 May 2022  23:54:34 +0300 (0:00:00.050)          0:10:18.069 *****
skipping: [wsl]

TASK [core/users : Change Windows users' passwords]
*****
Wednesday 11 May 2022  23:54:34 +0300 (0:00:00.050)          0:10:18.120 *****
included: /mnt/c/GIT/demo/roles/core/users/tasks/windows.yml for wsl

TASK [core/users : Verifying regular accounts]
*****
Wednesday 11 May 2022  23:54:34 +0300 (0:00:00.101)          0:10:18.221 *****
ok: [wsl] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [core/users : Verifying DC accounts]
*****
Wednesday 11 May 2022  23:54:39 +0300 (0:00:04.831)          0:10:23.053 *****
skipping: [wsl] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [core/users : Creating a profile for required accounts]
*****
Wednesday 11 May 2022  23:54:39 +0300 (0:00:00.071)          0:10:23.124 *****

TASK [core/users : Creating a profile for required accounts]
*****
changed: [ws2]

TASK [core/users : Creating a profile for required accounts]
*****
ok: [wsl] => (item={'username': 'Administrator', 'password': 'Password.123'})

TASK [Start VM customization]
*****
*
Wednesday 11 May 2022  23:54:41 +0300 (0:00:02.185)          0:10:25.309 *****

```

```
TASK [conf/join_domain : Adding Windows machines to domain]
*****
Wednesday 11 May 2022  23:54:41 +0300 (0:00:00.172)      0:10:25.482 *****
included: /mnt/c/GIT/demo/roles/conf/join_domain/tasks/windows.yml for ws1
```

```
TASK [conf/join_domain : Adding machine to domain]
*****
Wednesday 11 May 2022  23:54:41 +0300 (0:00:00.118)      0:10:25.600 *****
changed: [ws1]
```

```
TASK [conf/join_domain : Rebooting machine]
*****
Wednesday 11 May 2022  23:54:45 +0300 (0:00:03.986)      0:10:29.586 *****
changed: [ws1]
```

```
TASK [vm/ws1 : Add WS_RDP_Users to RDP group]
*****
Wednesday 11 May 2022  23:55:35 +0300 (0:00:50.136)      0:11:19.723 *****
changed: [ws1]
```

```
PLAY RECAP
*****
*****
```

```
localhost      : ok=1   changed=0   unreachable=0   failed=0   skipped=0
rescued=0      ignored=0

ws1             : ok=39  changed=10  unreachable=0   failed=0   skipped=4
rescued=0      ignored=0

ws2             : ok=38  changed=9   unreachable=0   failed=0   skipped=4
rescued=0      ignored=0
```

```
Wednesday 11 May 2022  23:55:43 +0300 (0:00:07.693)      0:11:27.417 *****
```