BACHEL DISSERTATION

How to Reduce the Time Needed to Create Firing Ranges?

Building a Fictitious Business Network in No Time

Bachelor	Applied Informatics
Elective Course	Computer & Cyber Crime Professional
Academic Year	2021 - 2022
Student	Maarten De Meyere
Internal Supervisor	Jill VandenDriessche (Howest)
External Promotor	Patrick Eisenschmidt (NVISO)

howest.be

BACHELOR DISSERTATION

How to Reduce the Time Needed to Create Firing Ranges?

Building a Fictitious Business Network in No Time

Bachelor	Applied Informatics
Elective Course	Computer & Cyber Crime Professional
Academic Year	2021 - 2022
Student	Maarten De Meyere
Internal Supervisor	Jill VandenDriessche (Howest)
External Promotor	Patrick Eisenschmidt (NVISO)

howest.be

Admission to loan

7/06/2022

Preface

The writing of this thesis means that my applied computer science course at Howest University of Applied Sciences in Bruges has come to an end. This work presents the result of a 15-week internship at NVISO in Frankfurt. This internship has given me an enormously interesting project that allowed me to give an equally interesting subject to this work. Therefore, I want to thank Nico Leidecker, Senior Manager, for assigning this project to me.

I also want to thank Patrick Eisenschmidt, Senior Cyber Security Consultant and my supervisor, for the guidance throughout the development process and the tips in relation to writing this thesis. Further, I want to thank Pierre-Alain Mouy, Managing Director, to allow this internship to happen and to provide me accommodation throughout my stay in Frankfurt. Naturally, I want to thank all employees I learned to know for the pleasant company and the fun activities.

For the feedback received from school, I would like to thank Jill VandenDriessche, my internship mentor and lecturer. To end things of, I want to thank my girlfriend, Mira Vogelsang, to provide me everyday with support, even from far away, and for filtering out some errors in this thesis.

I wish you, the reader, a pleasant day, and thanks for reading!

Maarten De Meyere
Bruges, 07/06/2022

Samenvatting

Een *firing range* is een virtuele omgeving dat meestal wordt gebruikt door pentesters en leden van een *red team* om hun vaardigheden te onderhouden of om onbekende of zelf gemaakte tools op uit te testen voordat ze worden gebruikt op een echt netwerk. Zo een virtuele omgeving stelt meestal een fictief bedrijfsnetwerk voor met daarin intentionele kwetsbaarheden verwerkt. Het is dan aan de pentesters om die kwetsbaarheden te ontdekken en aan de *red teamers* om ze te exploiteren door gebruik te maken van hun vaardigheden en tools in een veilige omgeving. Net zoals een echte schietbaan.

Het aanmaken van zo'n firing range is vaak een hele onderneming en neemt veel tijd in beslag. Wanneer deze dan klaar is om te gebruiken, is het beheren van elke machine opnieuw een omslachtige taak. Beheren gaat dan over plannen of een firing range relevant blijft en dus moet blijven bestaan en het bijhouden van een overzicht van wat de bedoeling is van elke firing range en welke rol elke machine daarin speelt. Een goede oplossing zou zijn om een applicatie te hebben waarin meerdere testomgevingen makkelijk kunnen worden gemonitord, aangemaakt, aangepast of afgebroken worden met slechts een paar toetsaanslagen in een paar minuten tijd. Zo een applicatie is een *firing range builder* wat direct ook het onderwerp is van deze thesis.

In deze zal worden nagegaan aan welke eisen moet worden voldaan om aan de behoeften van de beoogde gebruikers te voldoen, hoe de opzet en structuur van de applicatie eruit moeten zien, welke reeds bestaande hulpmiddelen een rol kunnen spelen in deze applicatie en wat de processen zijn achter de API-calls die zullen worden gebruikt om te communiceren met de server waarop deze nieuwe applicatie zal draaien.

Sleutelwoorden: virtuele machines, automatie, VMware vSphere suite, app ontwikkeling, firing range, testomgeving, Terraform, Ansible

Abstract

A firing range is a virtual environment that is mostly used by pentesters and members of a red team to maintain their skills or to assess unknown or self-made tools before these are used on a live network. This virtual environment commonly represents a fictitious business network with intentional vulnerabilities included. It is then the job for the pentesters to discover these vulnerabilities and for the red teamers to exploit them using their skillset and tools in a safe environment. Much like a real firing range.

Creating such a firing range is often a huge undertaking and takes a lot of time. When it is ready to use, managing each machine is again a laborious task. Managing is then about planning whether a firing range remains relevant and therefore must continue to exist and keeping a record of what the purpose of each firing range is and what role each machine plays in it. A great solution would be to have an application in which multiple testing environments can easily be monitored, created, edited, or destroyed in just a few clicks and in a few minutes time. Said application is a firing range builder which this bachelor dissertation will be all about.

This essay will consider what requirements need to be met to fulfil the needs of the target users, what the setup and structure of the application should look like, which already existing tools can serve a purpose in this application and what the processes are behind the API calls that will be used to communicate with the server on which this new application will run.

Keywords: virtual machines, automation, VMware vSphere suite, development, firing range, test environment, Terraform, Ansible

Abbreviations

API	Application Programming Interface
AD	Active Directory
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DB	Database
DC	Domain Controller
DHCP	Dynamic Host Configuration Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
OS	Operating System
PC	Personal Computer
SSH	Secure Shell
UI	User Interface
VM	Virtual Machine
WAN	Wide Area Network
WinRM	Windows Remote Management
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language

Glossary

Active Directory:	“A hierarchical structure that stores information about objects on the network.” (Iainfoulds et al., 2022)
Application Programming Interface:	The software through which multiple programs can interact with each other.
Domain Controller:	A server that oversees authentication and verification of users that want to access the domain’s resources and enforces security policies (Adjei, 2020).
Domain:	A controlled computer network in which registered users have access to information in a database (Stegner, 2018). In a Microsoft environment, this database would be an AD.
Dynamic Host Configuration Protocol:	A computer protocol that is used to exchange network information between the DHCP server and client.
Environment:	Network of interconnected devices.
ESXi Portgroup:	The VMware naming for a group of virtual interface ports bundled together to apply rules on them (Wahl & Pantol, 2014).
Firewall:	“A network security device that monitors and filters incoming and outgoing network traffic based on an organization’s previously established security policies.” (Check Point, n.d.)
Firing Range Builder:	An application to manage and create different firing ranges.
Firing range:	Virtual environment with intended vulnerabilities to test skills and tools.
Guest:	In terms of virtualization, this is the virtual machine running on a host.
Host:	In terms of virtualization, this is the physical machine on which VMs run.
Hypervisor:	A program used to run and manage one or more virtual machines on a computer.
ISO File:	“An exact copy of an entire optical disk such as a CD, DVD or Blu-ray archived into a single file.” (Naor, 2021)
JavaScript Object Notation:	A format which is used to interchange data between applications, and which is also easy to read for humans (*JSON*, n.d.).
Local Area Network:	A network of limited size that connects computers and other devices in a way they can communicate with each other. For example, a home network.
Operating System:	The software that performs all low-level tasks of a computer and on which can be interacted with other programs.
Pentest:	Or penetration testing is a procedure where the security of a network is tested, and vulnerabilities are documented.
Scenario:	A plot for a firing range that describes a fictitious business network.

Secure Shell:	A program that enables encrypted communication between server and client.
Suite:	A collection of computer programs with related functionality and a similar user interface that can easily exchange data (Wikipedia-Contributors, 2020).
User Interface:	The software or hardware through which a user interacts with an application.
Virtual Machine:	A compute resource running its own operating system while only using a part of the total amount of resources so that multiple virtual machines can run on one piece of physical hardware (VMware, n.d.).
VMware vCenter Template:	A model virtual machine which serves as a base to create other VMs. This is a VMware vCenter feature.
Wide Area Network:	A large network consisting of multiple LANs that can communicate with each other (Wikipedia-Contributors, 2021). For example, the internet.
Windows Remote Management:	A protocol by Microsoft that enables encrypted communication between hardware and operating systems (Stevewhims et al., 2021).

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This section contains a summary for whom this dissertation might be intended, how it came into existence, which problem is set out to be solved and how to proceed into resolving it.

1.1 In General

The topic covered is primarily meant to help the cybersecurity industry save time with setting up testing environments but can also be used within any domain that seeks for a solution to quickly setup and manage virtual environments. The solution in its final form will be a program which is easy to use by both experts and aliens in the trade (e.g., a person in HR can build an environment to evaluate a new employee).

The occasion for the research in this essay is a 15-week internship (14 February 2022 – 25 May 2022) at NVISO, a cybersecurity company that delivers various services to other businesses. These services fall under the prevention, detection and responding to cyber security related issues.

NVISO sought for a way to improve their method of operating with virtual machines and testing environments to save time and other resources. This paper provides a solution.

1.2 Problem Statement

The problem to be addressed is as follows:

For a cybersecurity business, it is imperative to be able to assure its clients of no downtime of their systems during a pentest. Therefore, the pentesters need to test their tools and skills on enough virtual testing environments to guarantee their reliability.

The way to go up until now is to manually create one big environment which exists out of sometimes tens of virtual machines which need to be configured to exercise some role in the network. It is understood that creating a business network, fictitious or not, requires a lot of time, let alone to include vulnerabilities in a strategic manner.

Creating a firing range that includes multiple real life network situations is impractical and hard to manage. Hence, it would be useful to have more than one shooting range at one's disposal. These could all be manually created which, again, takes a lot of time but also, as an additional problem, requires a lot of storage, even though not all environments need to be always available.

1.3 Research Question

The problem statement above can be summarized into one short research question:

How to reduce the time needed to create firing ranges?

This question can be divided into several sub-questions:

- Can the manual process be improved or is automation required?
- What already existing tools are useful for this project?
- Can the creation of a firing range be simplified so that a novice can perform it?
- In an on-premise situation, which hypervisor is best suited?

It is to these questions that an answer will be given by drawing up an experiment and processing the results.

1.4 Experiment

Unclear parts in this section may become clear after reading section 2.1.4.

After the test rig has been set up, tests will be conducted in which the same, small-scale firing range needs to be built in the following three ways:

- Manually beginning from ISO files. (Creating VMs, initial OS setups, configurations)
- Manually beginning from vCenter templates. (Creating VMs, configurations)
- Automatically using the firing range builder. (Using Terraform and Ansible)

The test firing range needs to contain the following machines with matching configurations:

Table 1: Test Firing Range

Machines	**Terraform Configurations**	**Uploaded / Ansible Configurations**
OPNsense router	/	• Custom LAN IP address • Accordingly adapted DHCP settings • Allow private and bogon addresses through WAN • Firewall rules: ○ WAN full access to LAN ○ LAN full access to all but WAN ○ Network of choice has full access to LAN
Windows Server 2022	• Static IP • Custom hostname • Custom admin password	• Install AD • Promote to DC
Windows 10	• Dynamic IP • Custom hostname • Custom admin password	• Domain joined • Firewalls disabled
Debian 11	• Dynamic IP • Custom hostname • Custom root password	• Install Apache2 • Webpage saying, “Hello World!”

During the three setups, times are recorded to indicate which role templates and automation play in reducing setup times. Therefore, two kinds of configurations are timed: Those performed by Terraform and those by Ansible. The Terraform infrastructure creation and configurations are executed in one go and are thus put together in one time. Ansible configurations are executed separately and are thus timed per machine. The manual setups are timed in such a way that first all VMs must be created and configured as if they were made by the Terraform process. Then, new times are taken per machine to finish the configurations as if performed by Ansible. Note that the used templates include additional configurations to make the automatic setup possible, but these do not need to be included in the manual tests since those configurations are rendered moot in a manual setup process.

Be aware that test results will differ according to which hardware is used to run all programs. The test rig in this case exists out of one physical machine with following specifications:

- Model: PowerEdge R6515 by Dell Inc.
- CPU: AMD EPYC 7402P 24-Core Processor
- Memory : 63.48 GB

The kind of experiment described is experimental and of quantitative nature as time-based numbers are compared from which conclusions are made. A part is also of qualitative nature as the experiences during the process of creating the experiment are shared (Merkus, 2022). The data collection method for this essay is field research as useful data is generated from an original experiment and quantitative observations (Benders, 2021). The method of data analyses is quantitative as numerical values are used to research the causal relationships between variables (Benders, 2021).

1.5 Structure of This Bachelor Thesis

This dissertation starts with what the experiment entails and how it can be reproduced. First, the requirements of the experiment are described. They explain what needs to be developed on behalf of the research topic and which features need to be included. Then, pre-existing solutions that can contribute to the project are laid out. After merging the requirements and existing solutions, a new solution is created: the firing range builder. An explanation follows about the workings of the solution and a guide to reproduce the experiment and tests.

After the tests have been conducted, reflections are made on how the research progress went and what differed from the research design, whether sufficient variables were considered, what the role of the researcher was, how the research situation turned out, and how the data was analysed.

With all useful data gathered, the results are disclosed, and the sub-questions are answered with the corresponding data visualised. This concludes the experiment.

Then, a short theoretical framework is presented with existing ideas and theories that are relevant for this research.

The thesis ends with a conclusion, discussion, and reflection where the research question is answered, the hypothesis is refuted or confirmed, results are interpreted, a conclusion is drawn as to whether this investigation was valid, and a reflection is made on what worked and what could have been better during this project.

2 Experiment

This section describes the experiment in detail and justifies the choices made. Further, the results are discussed by means of critical analysis.

2.1 Methodology

Since the cause of the problem is the manual labour required to set up an environment, the solution consists of process automation, as this removes human input. Process automation boosts quality, decreases errors, but most of all, it decreases the time required to deliver a result.

2.1.1 Requirements

The problem statement and the need for process automation call for an application that can build an entire, working network environment, completely autonomous with a single push of a button. The cyber world labels such an application a *firing range builder* as it builds multiple environments in which tools and skills are *fired* for testing purposes. These environments are called *firing ranges*. Following requirements have been presented during an interview (see section 3.2) to check their viability.

A configuration is needed to describe what the firing range should look like. Such a configuration is called a scenario. A scenario should include the components that make a network environment work: subnets, machines, and the configuration of those machines.

The primary goal of this application is to spend as little time as possible creating firing ranges. This is the case for both this research as for real world uses. But for this application to become a relevant day-to-day tool for consumers, it needs more features than the *create* action alone. The firing range builder should additionally be able to store scenarios for future use and remove or edit them once deprecated. To save computing resources, machines in a firing range should be able to be halted and started again. Finally, destroying firing ranges is required to save storage space. This way, the application becomes more of a firing range manager than a builder alone.

Developing a system that makes use of these qualities can create and destroy the same complicated environments repeatedly using the same, saved scenario. Succeeding in implementing these requirements would not only save an enormous amount of time but would also save resources of the server running the environments.

The only thing left to consider is which platform to use as host to build the firing ranges onto. The choice between on premise or in the cloud needs to be made. As of late, cloud computing is increasingly becoming the better option because of its flexibility and the worry-free IT, but as NVISO, the place this experiment was executed, has a more than capable on-premise server, it was decided to use that server as host for the firing ranges.

2.1.2 Pre-Existing Solutions

Currently, there does not seem to exist a public application which complies with the needed requirements. There are, however, applications that match with individual requirements.

Hypervisors

A specific kind of program is needed to run VMs onto, such a program is called a hypervisor. Hypervisors are the backbone of the creation and management of virtual machines. Out of the box, they are great to manually configure individual VMs, but are not ideal to set up large environments. Luckily, there are applications that can automate this process.

More information about Hypervisors can be found in the Theoretical Framework section.

Packer (HachiCorp, n.d)
Packer by HashiCorp is a tool used to build custom machine images locally or for cloud providers. It can be used to automate the process of creating custom ISO files or templates. At first, this seemed like a useful tool for this project but later, that idea was left behind since it was not a priority to automate template creation. Templates only need to be made once which does not necessarily need automation.

Terraform (HashiCorp, n.d.)
Terraform by HashiCorp is a tool that is designed to manage large infrastructures both local and remote in the cloud. Some of the providers supported are AWS, Azure, Google Cloud Platform, Kubernetes, VMware, etc. A file defines which provider to use and what machines need to be created with which resources and basic configurations. Once everything is specified in this file, a command is needed to initialize the setup. This installs the required packages to communicate with the provider. A next command builds the described infrastructure in a matter of minutes.

Therefore, Terraform is suited to oversee the creation of machines in firing ranges.

Ansible (Red Hat, n.d.)
Another tool deemed useful is Ansible. A program by Red Hat that, by means of a *playbook*, can apply configurations over multiple machines at the same time. Such a playbook describes what actions Ansible needs to execute on the remote machine.

These Ansible playbooks can serve as *features* for machines. For example: A fresh Windows Server 2022 has been created by Terraform. To add purpose to this machine, it needs to be promoted to a domain controller. So Ansible connects to this machine and executes the playbook in which is stated that the machine needs to become a domain controller. In a similar fashion, playbooks can be used to upload files, install software, edit settings, and so on. They only need to be created once and can then be selected to be used on multiple machines.

2.1.3 New Solution

Since no existing application satisfies the requirements and thus the needs for this research, a new one needs to be developed. The new solution is a server which uses a combination of Terraform, Ansible and original ideas to orchestrate firing ranges. The server itself is a Node.js application written in TypeScript and communicates with a client, which is not in scope for this project, via a REST API in JSON. Scenarios and firing range specific data are stored on a MariaDB database. The data transfer between the database and the server is handled by TypeORM, a Node.js package.

The reason for the client-server approach is the flexibility that comes with separating front- and backend. Node.js makes for an easier development progress, especially with asynchronous events, and supports a wide scheme of community made packages (Dziuba, 2021). TypeScript is chosen above JavaScript because it is an object-oriented language which supports classes, inheritance, namespaces, typing, etc. These characteristics make for a language that is better suited for large scale, server-side development (Raval, 2022).

As mentioned in the requirements, a capable, on-premise server, dedicated to run VMs, is available. Therefore, the VMware vSphere suite has been selected to manage and accommodate the VMs. This virtualization platform was chosen because VMware is currently the market leader (Angelino, n.d.) and the suite provides particularly useful features that will help reach the goal of this experiment. More about these technologies in the next topic.

2.1.4 How the Firing Range Builder Works

Before anything can happen at all, the firing range builder needs to know what to do. Instructions are given by means of a *scenario*. This scenario describes which machines need to be present in the firing range with what configurations and modifications. Naturally, the builder cannot just create a personalized Windows machine out of nothing. Hence, two libraries are in place to provide the builder with the necessary resources to start from: *templates* and *features*. Templates define the machines that the builder can provide. They are a feature within vCenter and are essentially VMs that serve as a base to build adapted versions from. Features are the customizations that can be made on VMs in the form of Ansible playbooks with variables to provide per firing range customization. These libraries are the key components that make the builder as effective as it is.

The next issue is the impossible task to provide every modification for every machine in existence. Therefore, it is up to the user to create the necessary templates and features for their needs. Fully adapting the builder to the personal needs can take some time, but once this one-time process is done, production speed will skyrocket. This modularity is the application’s trump card.

The firing range builder can be seen as a big Terraform and Ansible wrapper application. The scenario it gets from user input is on the one hand converted into a Terraform build plan, and on the other into multiple Ansible playbooks. This is done using Handlebars templating for Terraform and filling in the variables from Ansible playbooks before they are run.

Access to the VMs is required for the builder to change their configurations. Therefore, each template has the same extra user next to the usual ‘Administrator’ or ‘root’ users which is only used by the builder to execute Ansible playbooks. This extra user has admin privileges to the machines and must have a strong, unbreakable password as it should not be part of any testing.

First, Terraform is run which creates the required VMs from vCenter templates with some basic configurations like CPU count, memory, administrator password (only Windows), network and hostname. Once all VMs are up and running, the separate Ansible playbooks are executed in the order provided by the scenario. With this done, a simple, plaintext scenario has been converted into a useful, virtual playground.

With the described way of working, having multiple firing ranges would lead to problems where VMs from one range could communicate with VMs from another. Also, defining a scope to pentest would be cumbersome. Therefore, each firing range needs to reside in a segregated network from the others. To accomplish this, unique network addresses, maximum two subnets, are assigned to each firing range, and an OPNsense router is added by default to each scenario. Just like the other machines, this router is added into the Terraform build plan and added to the firing range from a template. As next step, before the features are executed, an adapted configuration file, through templating, is uploaded to the router with the custom network settings of that firing range.

During the time that the firing range exists, the Terraform files are kept because they are re-used when the range needs to be destroyed. One Terraform command and all the VMs of that environment are destroyed.

Annex 1 gives an overview of all behind-the-scenes process of all actions the application can execute. Annex 2 is a component diagram of how each element works together.

2.1.5 Environment Preparation

The vSphere suite includes two pieces of software: ESXi, a type 1 hypervisor, and vCenter Server, a central management system for ESXi hosts (Mayer, 2020). vCenter is normally unnecessary in case only one ESXi system is in use, but seen that vCenter complements ESXi with essential features, it is included in the project anyway. The extra features that make vCenter play such a significant role are the templates that render VM creation much faster, the better equipped package within Terraform, and the more extensive API which provides more information about VMs.

The simplified figure below visualises which role each element has:

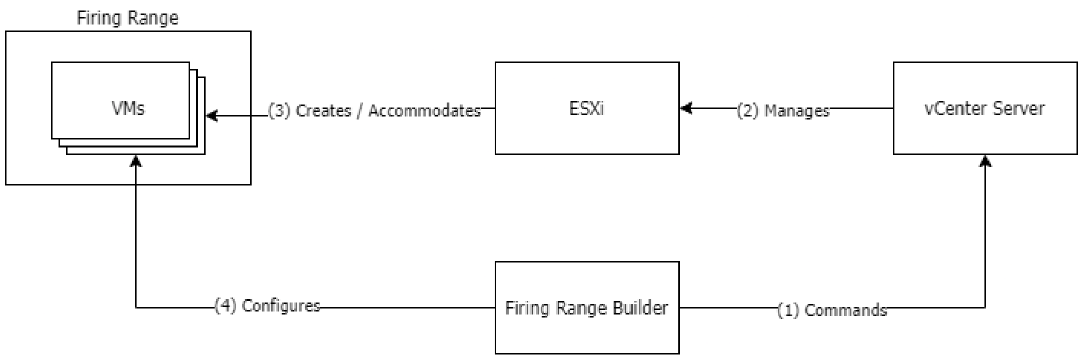


Figure 1: Systems in the Environment and Their Roles

Both ESXi and vCenter Server must be set up before the firing range builder can be used. In doing so, ensure that the networking part allows the builder to communicate with vCenter, ESXi and the VMs. Additionally, a DHCP server needs to be in place to provide new VMs their IP addresses, and SSH must be enabled on the ESXi.

Since the original environment for this experiment was created on a single physical machine, the general setup from scratch for this particular case is explained, without going into detail, in annex 3.

2.1.6 Set Up of the Firing Range Builder

This subtopic offers the required information about the steps required to set up the firing range builder, assuming that the preparations in the past subtopic have been made, so that the experiment may be reproduced. The actual roadmaps to follow per step can be found in the appendices. All software quoted here or in the appendices refers to the software used during the experiment. This does not mean that the experiment cannot be reproduced using other configurations.

The source code is available on https://gitfront.io/r/MartienskieNum1/1wTDf71Py6KE/BAP-Firing-Range-Builder/. This is the exact code that has been written during the time span of the internship apart from some solved bugs afterwards. Note that this project is being further developed by employees at NVISO. A more mature version can thus be found on https://github.com/NVISOsecurity.

System Requirements
First, a machine needs to be prepared on which the firing range builder and other necessary software must be installed. This machine, virtual or not, needs to be able to communicate with the vCenter Server, ESXi and VMs. All software requirements and steps to follow to prepare this machine can be found in annex 4.

Configuration File
Then, the builder needs to be configured so it knows where to find all resources and how to access them. This is done by filling in the variables in the configuration file. A list of these variables and their declaration is in annex 5.

Database
With the builder configured, the next step is to run it for the first time. No actions can be taken yet since no templates or features have been created, but this first run will create all necessary tables and columns in the provided database. The database scheme can be found in annex 6.

Templates
Next up, templates need to be created via the vSphere client and then added to the database. Creating a new template is not too complicated: Create a VM and complete the initial setup as regularly until the desktop is shown. Turn off the VM and convert it into a template. There are, however, extra steps to keep into account according to which kind of VM is being created: Windows or Linux. A roadmap for both can be found in annex 7 and 8 including the steps to insert the template into the database. Note that templates need to be placed in the 'Preinstalled Templates' folder and that the storage size of a VM will be the same of that of a template.

Default Router
There is one exception on the normal course with templates: The OPNsense router. This system runs neither Windows nor Linux as OS but uses OpenBSD instead, a Unix-based OS. Therefore, this template is customized by uploading a configuration file, instead of using Terraform or Ansible. A roadmap on how to set up this template and what the configuration file does can be found in annex 9.

Features
With the current setup, firing ranges can be created with basic Terraform configurations like networking, hostnames, and administrator passwords, but that does not make the builder special. The last thing to do to unlock the builder's full potential is to add features in the form of Ansible playbooks. Features enable the user to modify VMs as they wish with Ansible variables that are defined in the scenario. Note that also files can be uploaded when creating a scenario so that they can be used in a feature by inserting the file name in a variable. A guide on what to consider when creating one and how it is added in the database can be found in annex 10. Multiple examples can be found in the source code. One of these examples can be found in annex 11 as a guideline to create a new one.

2.1.7 Using the Firing Range Builder

User Interface

As this project was entirely focused on delivering a working prototype, no user-friendly front-end UI was developed. However, there is a Swagger UI in place which presents an interactive visualization of the API's resources (Swagger, n.d.). It presents all API endpoints with a clarification of what it does, what the parameters and request bodies should look like with an example, and which responses to expect. It is accessible on http://builders-address:3000/api-docs. For the functionality to work, add the address of the builder to the list of servers in the beginning of the Swagger specification file (`src/swagger.yaml`).

API
All API endpoints can be found in annex 12. The body schemes to use and their clarification are found in annex 13 and 14.

Reproducing the Tests
After all setup procedures are completed, the firing range builder can be used to conduct the automated test required to complete this research (see section 1.4) but to replicate this test is also the same input required. The exact request body (or scenario) used can be found in annex 15. This body is used in the *create scenario* endpoint of the API with no input files required. The necessary features are present in the source code, but the templates still need to be made. The required templates are a Windows Server 2022 with as name `win_2022`, a Windows 10 20H2 with name `win_10_20H2`, a Debian 11 with name `debian_11` and the OPNsense router template with name `opnsense`.

Access to Firing Ranges
There is currently one way to access the sandboxes. Use a PC or VM that resides in the WAN of the firing ranges and create routes to the networks of the range to which access is needed. This works since all default routers are configured to allow inbound traffic from WAN to LAN.

If access to a range is required from another network, modify the template file of the router to grant access from another network than the WAN. This process is explained in annex 9.

2.2 Course of Research and Data Analysis Method(s)

NVISO has a virtual environment in place that they have been using for a while. Keeping this environment up to date with the latest updates and contemporary software required too much time and resources. Therefore, NVISO wanted a tool that makes creating these environments faster and makes rolling out updated versions easier. As all employees at NVISO were occupied doing other jobs, they let an intern have a try at it. Here is where the role of the researcher comes into play.

The first few days consisted of browsing around the internet for existing solutions and useful software. After messing around with a bunch of tool combinations, the first VM with a custom configuration was created through automation. From this point on became the goal of this project clear: Spawning a bunch of custom VMs with a single push of a button. The tools used at that time were VMware Workstation Pro, Packer, Terraform and Ansible. Later, access was given to a system running ESXi and a vCenter Server so that further experimentation could be conducted on those platforms instead of Workstation Pro.

And so, the development of a tool that brings the functionalities of Terraform and Ansible together began. Initially, the researcher's preferred platform was Laravel because of the nature of his education, but when the importance of a client-server approach for such an application became known, the switch to Node.js was made. From then on, the course of events followed that of any development project: Bi-weekly stand-up meetings, meetings to discuss implementation choices and Trello boards with nice-to-haves, must-haves, doing and done. It went like this for the greater part of the internship.

When the end of the term began to approach, some extra tasks needed to be taken into account. An extensive documentation was required so that employees could take over the development and comprehend the inner workings of the program. This documentation is present in the wiki of the NVISO repository. Additionally, a live presentation was required to explain the purpose of this internship and to present the prototype to other employees whom this topic might concern.

It was only after the internship that the tests for this research were conducted. These tests are needed to provide evidence that this 15-week project contributes to the community. The scenario for the test firing range was composed and then built by the firing range builder. First, the time it took for Terraform to finish was measured. Then, the time it took for Ansible to finish modifying each machine. Now, the same end result needed to be manually achieved. Once from scratch from ISO files and once with the help of templates that were also used by the builder. During these manual builds, the same times are measured to reflect the configurations done as with Terraform and Ansible. These times are then compared to, on the one hand determine the effectiveness of automation in general, and on the other, the effectiveness of using templates.

2.3 Results

After the experiment has been set up and the tests have been conducted (see section 1.4), the results can be presented. These results plus context from previous chapters are used to answer the sub-questions to then answer the main research question in section 4. This topic only establishes the link between the sub-questions and their results.

All raw test results are inserted in the table below. The times are subdivided by building method and then by building method into actions performed by Terraform and actions performed by Ansible per machine. The time required to build the four templates simultaneously is also included.

There is data present that needs clarification. The automated build has been performed three times to get info about this process’ consistency. In general, the configurations are consistent except for that of the Windows Server 2022. The inconsistency here is the process that creates a domain after the AD has been installed. Out of experience is found that interacting with the VM in question during this process speeds up the process. The researcher’s hypothesis for this is that because of the interaction, the VM reports back to Ansible immediately instead of having a delay in case of no interaction.

Table 2: Test Results

Build / Conf	Templates	Manual (ISO)	Manual (Templates)	Automated (1st run)	Automated (2d run)	Automated (3d run)
Terraform	/	00:34:26	00:13:12	00:07:17	00:07:12	00:07:31
Router		00:07:26		00:00:35	00:00:34	00:00:35
Windows Server 2022		00:10:35		00:12:50	00:16:06	00:04:36
Windows 10		00:05:47		00:02:59	00:03:00	00:03:00
Debian 11		00:05:44		00:00:12	00:00:11	00:00:11
Total	**00:25:30**	**01:03:58**	**00:42:44**	**00:23:53**	**00:27:03**	**00:15:53**

2.3.1 Can the manual process be improved or is automation required?

To answer this question, the times of the manual setup processes need to be compared to see whether the use of templates has made any change in terms of reducing the time it takes to manually set up a firing range. Next to those results, the results of the fully automated setup can be used to see whether automation is required at all. Note that creating templates also takes time but this is eventually nullified.

The figure below visualises how each setup method impacts configuration times.

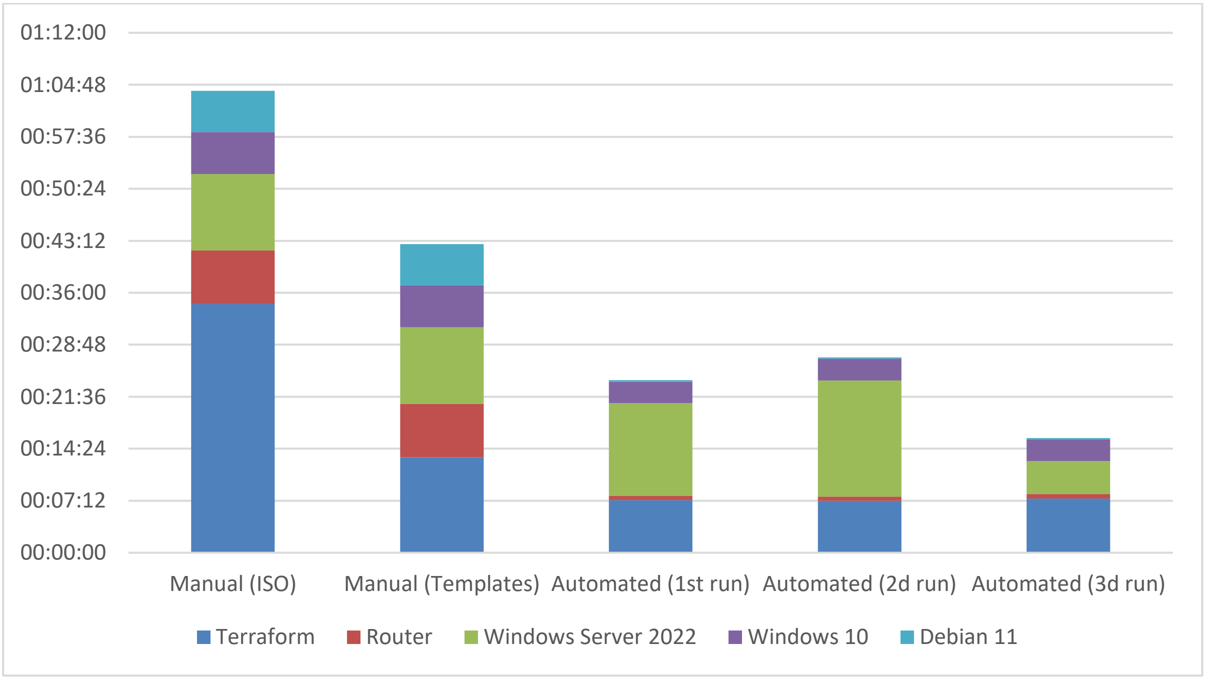


Figure 2: Setup Times Per Setup Method

2.3.2 What already existing tools are useful for this project?

Information and remarks to answer this question have been given in section 2.1.2 and section 2.1.3. In those sections is discussed why specific tools were used or not. Using the results from table 2, the effectiveness of the used tools can be expressed by percentages:

- vCenter templates improve manual VM creation time by 21 minutes and 14 seconds or 260,86 %. (34:26 – 13:12)
- Terraform (using templates) improves VM creation time by 27 minutes and 14 seconds or 478,24 %. (34:26 – 07:12)
- Ansible improves overall VM configuration time (after creation) by 21 minutes and 10 seconds or 352,99 %. (29:32 – 08:22)

2.3.3 Can the creation of a firing range be simplified so that a novice can perform it?

Section 2.1.7 explains how the firing range builder can be operated. The current methods in place require technical knowledge.

2.3.4 In an on-premise situation, which hypervisor is best suited?

The literature research about hypervisors (see section 3.1) provides the necessary information to answer this question.

3 Theoretical Framework/Literature Research

This section gives a critical overview of what already exists or is known about this topic. It serves to find out which gaps needs to be filled to contribute to this topic.

3.1 Hypervisors

Hypervisors are the software on which virtual machines can run. They are included with wizards and APIs to set up VMs and afterwards manage them.

There exist two types of hypervisors as shown in the figure below (ResellerClub, 2021):

Type 1 is called a bare-metal hypervisor. This is a program running directly on the physical hardware of the machine and acts as an OS with as sole purpose to manage VMs. The fact that it has direct access to the hardware makes this type the best performing, most secure and most efficient hypervisor. Some examples are VMware ESXi, Microsoft Hyper-V, and Citrix.

Type 2 is a hosted hypervisor. This means that the hypervisor is a program which has been installed on an already existing OS, like Windows. Because of this, the hypervisor endures more latency than type 1. Examples are VMware Workstation, VMware Fusion, and Oracle Virtual Box.

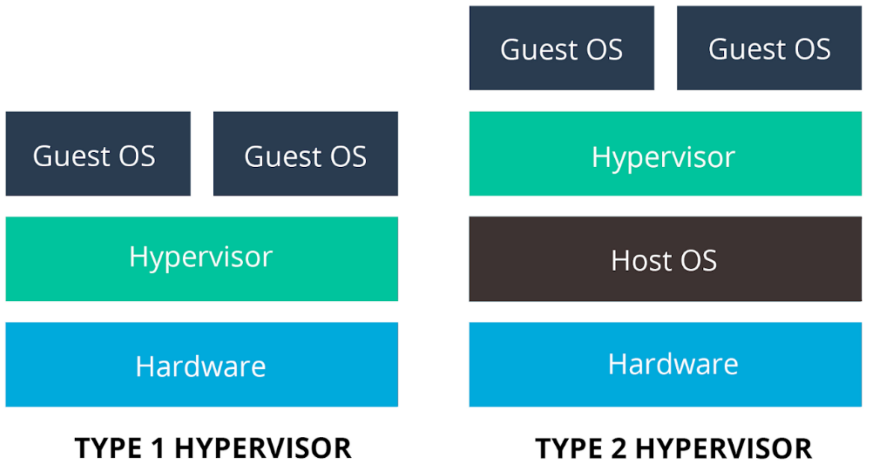


Figure 3: Difference Between Type 1 and Type 2 Hypervisors (ResellerClub, 2021)

3.2 Need for a Solution

An interview has been conducted with NVISO employees whom this project might concern. This interview served to get informed about the requirements needed for an application like the firing range builder. During the interview, remarks were made that such a tool would be important for the company and other companies alike. This, together with the lack of presence on the internet, suggest that the problem at hand is a common one with no known solution, at least not publicly available.

4 Conclusion

Now that the topic has been thoroughly addressed and tests have produced the necessary data, a conclusion can be formed to answer the opening hypothesis: How to reduce the time needed to create firing ranges? Before an answer can be given on this question, other questions need to be answered first.

Creating firing ranges is currently a manual task. So, is it possible to improve the current way of working? There is! VMware vCenter offers a feature, templates, from which new, ready to use, VMs can be created without the need of going through the initial setup process. From there, the time it requires to do custom configurations needs to be reduced. To achieve this, automation is needed.

The needs call for an application which automates the entire process of building firing ranges. Does a tool currently exist to solve this problem? No, but tools exist that can partially solve the problem. Terraform can create the VMs needed using the templates, and Ansible can afterwards modify each machine. Now, an application needs to be developed that combines the capabilities of both tools. That application will orchestrate the creation and modifications of VMs on a particular hypervisor, namely VMware ESXi. This type two hypervisor is a necessity to use the template feature of vCenter and is besides that a fast and good working option.

This new application, the firing range builder, only requires a scenario, a description of what the firing range should look like, and it will build it fully automatically in a few minutes time. An additional advantage of this project is that having to only create and store such a scenario suffices for the same scenario to be created and destroyed repeatedly while using a minimum amount of storage. Its simplicity even enables unexperienced personnel to build complex test environments provided that a user-friendly UI is available.

All this proves that it is possible to reduce the time needed to create a firing range and, to answer the hypothesis, it is done using the firing range builder. The application this dissertation was all about.

5 Discussion

This section serves to discuss the general course of events during this research.

In the researcher's opinion, this research was a remarkable success. The results speak for themselves. The NVISO employees who showed interest were excited to get their hands on a finished version of this application.

Even though the results are quite remarkable, they were not too much of a surprise. Already early in the testing phases of Terraform and Ansible became clear that if a way could be made for those tools to work together, the result would be well worth the development effort. This is of course because a manual process has been fully converted into an automated process which have proven multiple times to be way more efficient. In this respect, the firing range builder is equipped with more features than just combining two existing tools alone. This prototype has the potential to grow as a tool that could help cyber and networking specialists everywhere.

6 Reflection

Despite the positive outcome of this research and project, there is still room for improvement. Because of the only limited development experience of my cyber security focused education, it is most likely that the source code contains a lot of unconventional approaches to making things work. Additionally, even with the security background, the prototype contains a few no-goes in terms of security like clear text passwords and no proper checks on user input. The program could also do with more pre-made features.

With that said, this prototype needs a lot of finishing touches before it can be used as a reliable tool (e.g., front-end, authentication, bug fixes, choice between multiple virtual platform providers).

Sources & Literature List

Adjei, M. (2020, 9 December). *What Is a Domain Controller?* Illumio. Accessed 30 May 2022, https://www.illumio.com/blog/domain-controller

Angelino, A. (n.d.). *Overview of main virtualization vendors - Introduction to Virtualization Technologies Course*. Cloud Academy. Accessed 31 May 2022, https://cloudacademy.com/course/introduction-to-virtualization-technologies/virtualization-vendors-overview-1/#:%7E:text=Nowadays%20VMware%20is%20the%20virtualization,to%20deploy%20full%20virtualized%20machines.

Benders, L. (2021, 8 November). *De methodesectie (methodologie) in je scriptie*. Scribbr. Accessed 30 May 2022, https://www.scribbr.nl/scriptie-structuur/methodologie/

Check Point. (n.d.). *What is a Firewall? The Different Types of Firewalls*. Accessed 30 May 2022, https://www.checkpoint.com/cyber-hub/network-security/what-is-firewall/#:%7E:text=A%20Firewall%20is%20a%20network,network%20and%20the%20public%20Internet.

Dziuba, A. (2021, 27 August). *7 Advantages of Node.js for startups*. Relevant Software. Accessed 28 May 2022, https://relevant.software/blog/7-benefits-of-node-js-for-startups

HashiCorp. (n.d.). *Build automated machine images*. Packer. Accessed 6 June 2022, https://www.packer.io/

HashiCorp. (n.d.). *Providers*. Terraform Registry. Accessed 27 May 2022, https://registry.terraform.io/browse/providers

HashiCorp. (n.d.). *Vagrant vs. Terraform*. Vagrant. Consulted on 24 May 2022, https://www.vagrantup.com/intro/vs/terraform

Iainfoulds, I., Dknappettmsft, V-kents, Eross-msft, DCtheGeek, Imba-tjd, JasonGerend, MicrosoftGuyJFlo, Billmath, & Sudeepku. (2022, 11 January). *Active Directory Domain Services Overview*. Microsoft Docs. Accessed 6 June 2022, https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview

JSON. (n.d.). json. Accessed 31 May 2022, https://www.json.org/json-en.html

Mayer, A. (2020, 19 February). *VMware ESXi vs vSphere vs vCenter: A Comparison*. NAKIVO. Accessed 31 May 2022, https://www.nakivo.com/blog/vmware-esxi-vs-vsphere-vs-vcenter-key-differences/#:%7E:text=vSphere%20is%20an%20industry%2Dlevel,with%20just%20a%20few%20clicks.

Merkus, J. (2022, 20 May). *Overzicht van onderzoeksmethoden en dataverzamelingsmethoden*. Scribbr. Accessed 30 May 2022, https://www.scribbr.nl/onderzoeksmethoden/soorten-onderzoeksmethoden/

Naor, A. (2021, 19 February). *What Is an ISO File? Explained in Plain English*. freeCodeCamp. Accessed 30 May 2022, https://www.freecodecamp.org/news/what-is-an-iso-file-explained-in-plain-english/#:%7E:text=An%20ISO%20file%20is%20an,of%20large%20sets%20of%20data.

Raval, N. (2022, 24 May). *TypeScript vs JavaScript: The Difference You Should Know*. Radixweb. Accessed 30 May 2022, https://radixweb.com/blog/typescript-vs-javascript

Red Hat. (n.d.). *How Ansible Works*. Ansible. Accessed 27 May 2022, https://www.ansible.com/overview/how-ansible-works

ResellerClub. (2021, 10 December). *Type 1 and Type 2 Hypervisors: What Makes Them Different*. Medium. Accessed 29 May 2022, https://medium.com/teamresellerclub/type-1-and-type-2-hypervisors-what-makes-them-different-6a1755d6ae2c

Stegner, B. (2018, 22 March). *What Is a Windows Domain and What Are Its Advantages?* Make Use Of. Accessed 6 June 2022, https://www.makeuseof.com/tag/windows-domain/

Stevewhims, Mattwojo, DCtheGeek, Drewbatgit, SagiHello, Mijacobs, & Msatranjr. (2021, 11 September). *Windows Remote Management - Win32 apps*. Microsoft Docs. Accessed 4 June 2022, https://docs.microsoft.com/en-us/windows/win32/winrm/portal

Swagger. (n.d.). *REST API Documentation Tool | Swagger UI*. Accessed 5 June 2022, https://swagger.io/tools/swagger-ui/

VMware. (n.d.). *What is a virtual machine?* Accessed 29 May 2022, https://www.vmware.com/topics/glossary/content/virtual-machine.html

Wahl, C., & Pantol, S. (2014). The vSphere Standard Switch. In *Networking for VMware Administrators* (1st edition, p. 11). Pearson Education. https://www.pearsonitcertification.com/articles/article.aspx?p=2190191&seqNum=11

Wikipedia-Contributors. (2020, 19 June). *Software suite*. Wikipedia. Accessed 31 May 2022, https://nl.wikipedia.org/wiki/Software_suite

Wikipedia-Contributors. (2021, 14 December). *Local area network*. Wikipedia. Accessed 30 May 2022, https://nl.wikipedia.org/wiki/Local_area_network

Overview of the Annexes

Annex 1: Process Diagram

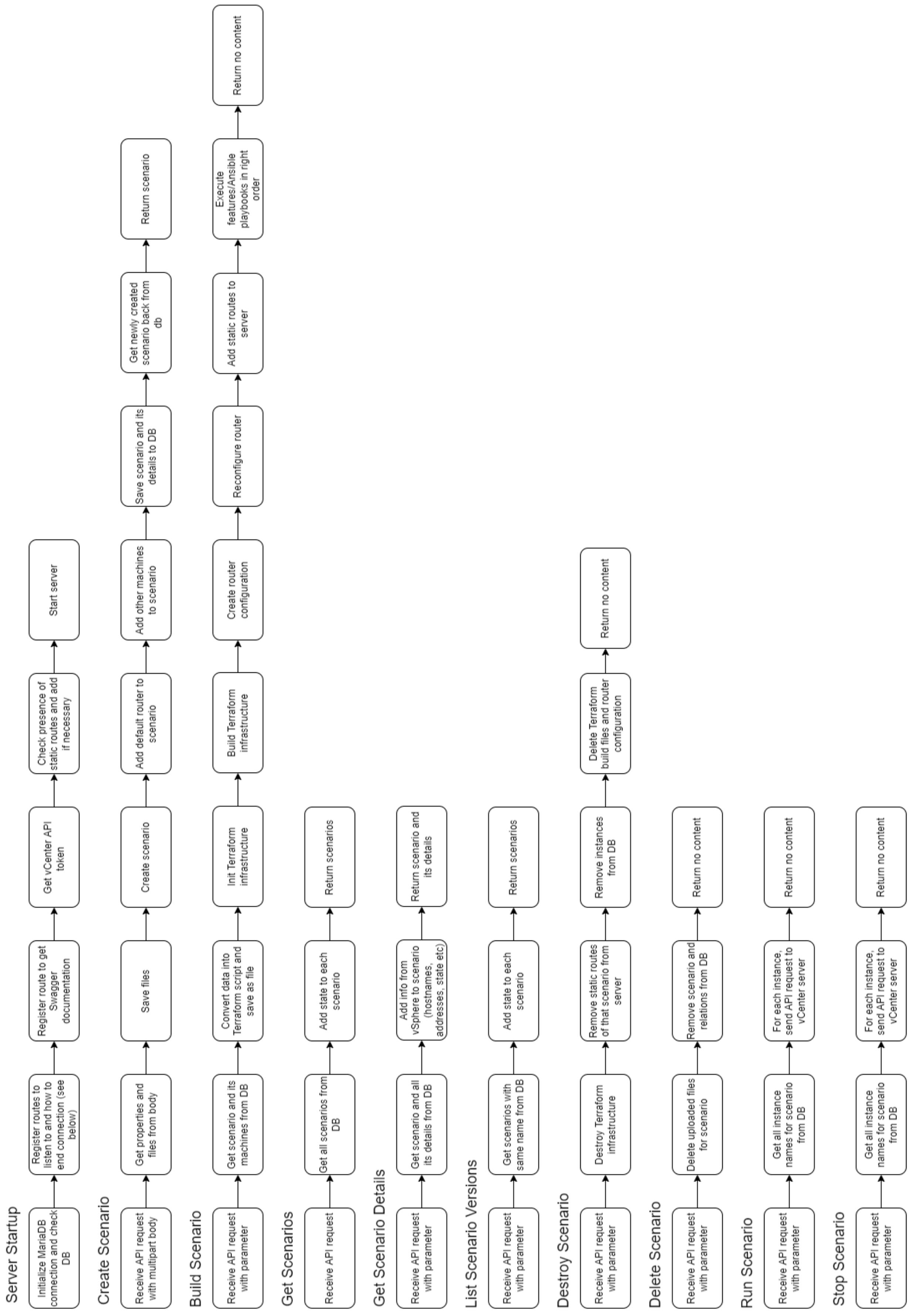

Server Startup
Initialize MariaDB connection and check DB
Register routes to listen to and how to end connection (see below)
Register route to get Swagger documentation
Get vCenter API token
Check presence of static routes and add if necessary
Start server
Create Scenario
Receive API request with multipart body
Get properties and files from body
Save files
Create scenario
Add default router to scenario
Add other machines to scenario
Save scenario and its details to DB
Get newly created scenario back from db
Return scenario
Build Scenario
Receive API request with parameter
Get scenario and its machines from DB
Convert data into Terraform script and save as file
Init Terraform infrastructure
Build Terraform infrastructure
Create router configuration
Reconfigure router
Add static routes to server
Execute features/Ansible playbooks in right order
Return no content
Get Scenarios
Receive API request
Get all scenarios from DB
Add state to each scenario
Return scenarios
Get Scenario Details
Receive API request with parameter
Get scenario and all its details from DB
Add info from vSphere to scenario (hostnames, addresses, state etc)
Return scenario and its details
List Scenario Versions
Receive API request with parameter
Get scenarios with same name from DB
Add state to each scenario
Return scenarios
Destroy Scenario
Receive API request with parameter
Destroy Terraform infrastructure
Remove static routes of that scenario from server
Remove instances from DB
Delete Terraform build files and router configuration
Return no content
Delete Scenario
Receive API request with parameter
Delete uploaded files for scenario
Remove scenario and relations from DB
Return no content
Run Scenario
Receive API request with parameter
Get all instance names for scenario from DB
For each instance, send API request to vCenter server
Return no content
Stop Scenario
Receive API request with parameter
Get all instance names for scenario from DB
For each instance, send API request to vCenter server
Return no content

Figure 4: Process Diagram

Annex 2: Component Diagram

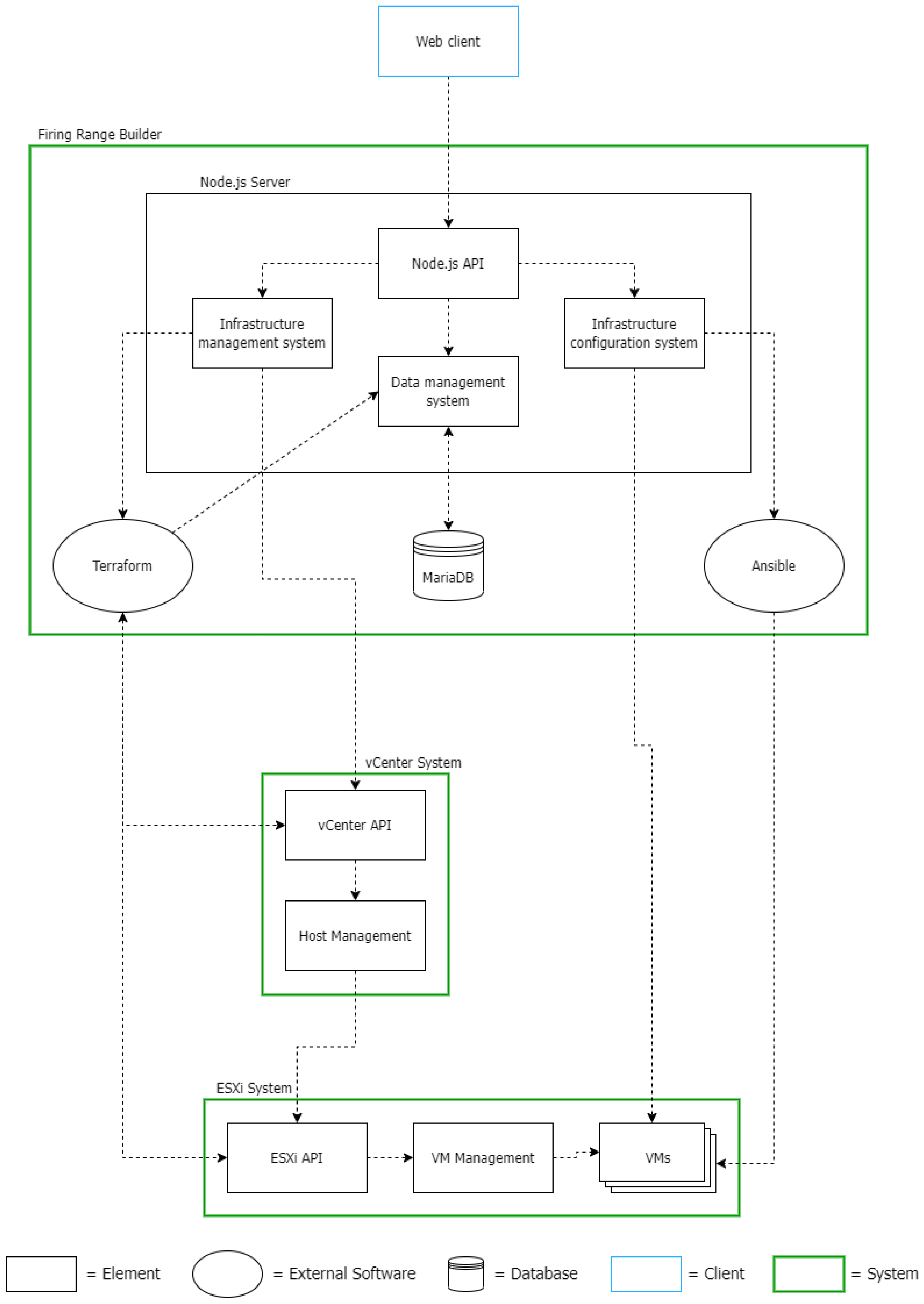
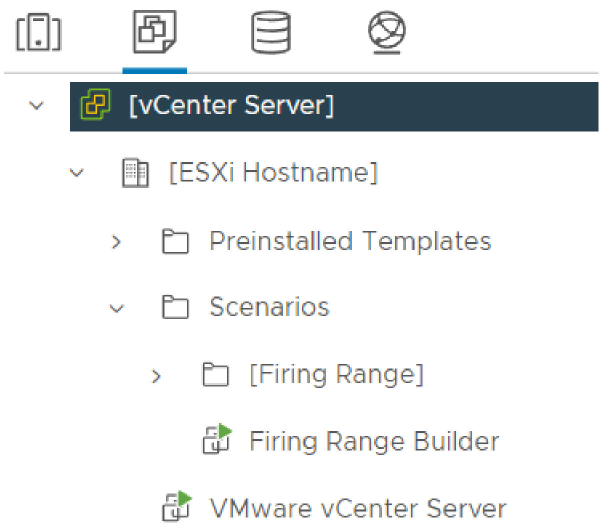


Figure 5: Component Diagram

First, ESXi must be installed as OS on the host (i.e., the physical device). Then, use a PC to surf to the address of the ESXi and create a Windows VM. On this VM, install vCenter Server. Surf to it, access vCenter through the vSphere web client, and add the ESXi host as one of its hosts. ESXi can now be managed by vCenter. Using the vSphere client, create a Debian VM and create two folders named ‘Scenarios’ and ‘Preinstalled Templates’ in the ‘VMs and Templates’ view. The former houses the firing ranges and in the latter reside the templates. Finally, clone the GitHub repository to the Debian machine.

In the sideview of the vSphere Client, the result should look something like this with varying values between the brackets:



```
[vCenter Server]
    [ESXi Hostname]
        Preinstalled Templates
        Scenarios
            [Firing Range]
            Firing Range Builder
        VMware vCenter Server
```

Figure 6: VMs and Templates View of vSphere Client

Operating System

Debian 11

Required Software

- Sudo
- SSH
- NPM
- Curl
- NVM (Node Version Installer to install specific version of Node)
- Node v17.8.0 (Other versions are not tested)
- Terraform
- Ansible
- Mariadb-server
- Mariadb-client (Make sure to run `sudo mariadb-secure-installation`)
- Sshpass

Post Installation Steps

- Clone the source code from the provided link.
- Setup database (local or remote).
  ```
  sudo mariadb
  create database [database name];
  create user '[new user]'@'%' identified by '[password]';
  grant all privileges on [database name].* to '[new user]'@'%';
  ```
- Remove the need for a sudo password for the `ip` command.

  Add `[user]   ALL=(ALL:ALL)      NOPASSWD:    /bin/usr/ip` to `/etc/sudoers`
- Navigate to the 'Server' folder and run `npm i`.
- Fill in the configuration file (annex 5).
- Start the server with `npm start`.

Annex 5: Configuration File

These are the variable names found in file `src/variables.tf` with their declarations:

- `vsphere_user`: Username of a vSphere user.
- `vsphere_password`: Password of the vSphere user.
- `vsphere_server`: IP address of the vCenter server.
- `vsphere_datacenter_name`: Name of the datacentre in which firing ranges will run.
- `vsphere_datastore_name`: Datastore on which to save all VMs.
- `vsphere_host_name`: Hostname of the host on which firing ranges will run. This is the ESXi server's name that resides in the set datacentre. If no name is given, it is the IP address.
- `vsphere_WAN_name`: This is the name of the network used as WAN. In most cases, it resembles an ESXi portgroup that first needs to be created.
- `esxi_user`: Username of an ESXi user.
- `esxi_password`: Password of the ESXi user.
- `esxi_server`: IP address of the ESXi system.
- `mariadb_user`: Username of a MariaDB user with sufficient permissions.
- `mariadb_password`: Password of the MariaDB user.
- `mariadb_server`: IP address of the MariaDB server. Is 127.0.0.1 if it is running local.
- `mariadb_database`: Name of the clean database to use.
- `scenario_router_user`: Username of an OPNsense user. Is "root" by default.
- `scenario_router_password`: Password of that OPNsense user. "opnsense" by default.
- `scenario_router_name`: The desired VM name for the router. Free of choice.
- `master_user`: Username of the global user which is present on each template.
- `master_password`: Password of the global user.

Annex 6: Database Scheme

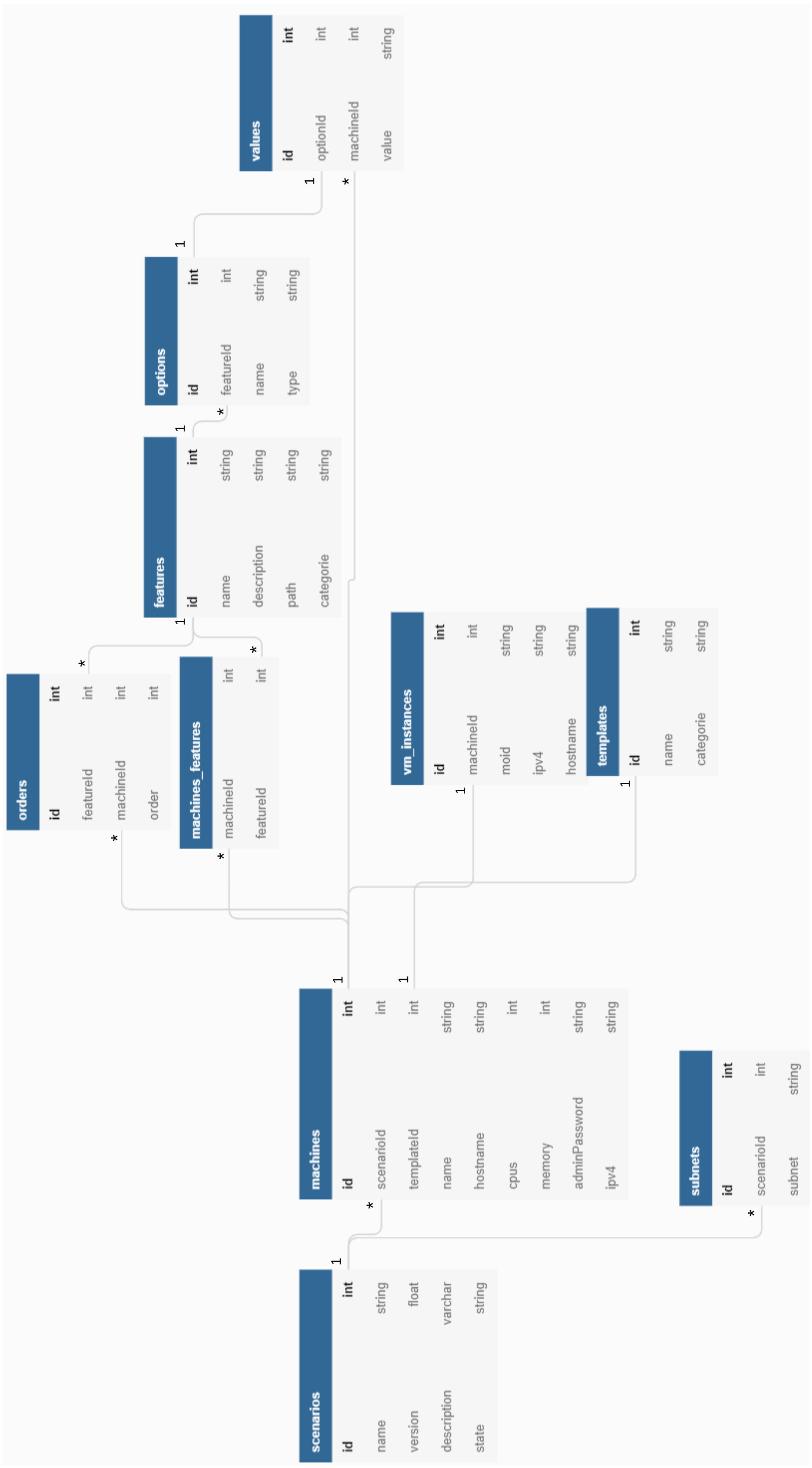


Figure 7: Database Scheme

Annex 7: Windows Template Roadmap

Following steps are required to create a Windows template that works with the firing range builder. Begin this process when the initial setup has been finished:

- Create the global user with its password (consistent across all templates).
- Install VMware tools (via the vSphere client).
- Make the machine pingable by changing the network environment from public to private and by enabling the inbound firewall rules "File and Printer Sharing (Echo Request - ICMPv4-In)".
- Disable sleep mode.
- Execute following commands in Powershell to enable WinRM. This protocol is used for communication with the builder:

```
Enable-PSRemoting -Force
winrm quickconfig -q
winrm quickconfig -transport:http
winrm set winrm/config '@{MaxTimeoutms="1800000"}'
winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="800"}'
winrm set winrm/config/service '@{AllowUnencrypted="true"}'
winrm set winrm/config/service/auth '@{Basic="true"}'
winrm set winrm/config/client/auth '@{Basic="true"}'
Set-Service winrm -startuptype "auto"
Restart-Service winrm
```

- Shutdown the VM.
- Convert to template.
- Add an entry in the `templates` table with as name the same name of the template and as category `windows`.

Annex 8: Linux Template Roadmap

Following steps are required to create a Linux template that works with the firing range builder. Keep some of these steps in mind when going through the initial setup process:

- When creating the VM, select as ‘Guest OS Family’ ‘Linux’ and as ‘Guest OS Version’ ‘Other Linux (64-bit)’.
- Root can be given a password at choice. Keep in mind that, different from Windows systems, the root user's password cannot be configured thru scenario composition.
- Install ‘sudo’.
- Install ‘SSH’.
- Create the global user with its password (consistent across all templates).
- Add the global user to the ‘sudo’ group.
- Shutdown the VM.
- Convert to template.
- (VMware tools should be installed by default)
- Add an entry in the `templates` table with as name the same name of the template and as category `linux`.

Annex 9: Router Template Roadmap and Its Custom Configuration

Following steps are required to create the OPNsense router template for it to work with the firing range builder:

- When creating the VM, select as ‘Guest OS Family’ ‘Other’ and as ‘Guest OS Version’ ‘FreeBSD 13 or later versions (64-bit)’.
- Name the VM ‘opnsense’.
- Root user's password is free of choice (READ NOTE)
- System/Settings/Administration (For uploading of new configuration)
    - Enable SSH
    - Permit root user login
    - Permit password login
- Services/DCHPv4/[LAN] (So that other machines get new addresses quickly)
    - Default lease time (seconds): 60
    - Maximum lease time (seconds): 61
- Shut down the VM.
- Convert to template.
- Add an entry in the `templates` table with as name the same name of the template and as category `default_router`.

The new router configuration is adapted to the custom subnet configuration defined in the scenario. Following changes are made in comparison with the default configuration:

- Interfaces/[NET]
    - IPv4 address: 10.x.x.1
- Interfaces/[WAN]
    - Disable "Block private networks"
    - Disable "Block bogon networks"
- Services/DHCPv4/[NET]
    - Range from 10.x.x.2 to 10.x.x.254
    - Default lease time: back to default value
    - Maximum lease time: back to default value
- Firewall/Rules/NET
    - Default rules are removed
- Firewall/Rules/Floating
    - WAN has full access to LAN
    - LAN has access to everything but WAN
    - Network of choice is allowed to access LAN
      Configure this network on line 546 of `src/templates/opnsense_conf.ts` and type a 1 on line 528 to enable that alias. The purpose of this is to allow machines that are not included in the WAN to access the ranges.

NOTE

The new configuration file is made according to the templates in `src/templates/opnsense_conf.ts`. A password hash is present in a template on line 221 which is the hash for "cOKstrvdr1Bnus9bAyEQ". This is a randomly generated password which each ‘root’ user will have in a firing range. In case this password is left unchanged for simplicity’s sake, it is recommended for the template to also have that same password in case the template is reverted to a VM to make modifications.

Should it be preferable to change this password: Set a password of choice in the template, download the configuration under System/Configuration/Backups, go look in that XML file for the hash of your password and copy paste it into the template on line 221.

Creating

This guideline provides recommendations and describes the process to create an Ansible playbook in YAML to be used as feature. It is recommended to use the provided example in annex 11 as a guideline.

The first task is performed on the builder itself and is the same for every feature. It temporarily adds the to be configured VM as host so that next tasks can be executed thereon. It also defines what protocol Ansible needs to use to communicate with the VM. This is either WinRM or SSH depending on whether that feature is meant for Windows or Linux based machines. Note that when defining hosts, `gather_facts` is set to `false` to not trigger unexpected errors.

All following tasks are executed on the VM and define what the feature's purpose is. It is advised that the first task executed on the VM is `wait_for_connection` to make sure that the machine concerned is reachable before continuing although the builder first checks this.

As features need to be dynamic to work with multiple scenarios, not all values can be hard-coded into the feature. Therefore, flexible values may be replaced by variables which can be defined in scenarios. The naming of those variables is mostly by choice but note that underscores are ought to be used instead of spaces or dashes as such: `{{ its_a_variable }}`.

Note that some variable names are reserved and used by the builder to provide Ansible details about the machine to which the user has no access:

- `{{ address }}`: IPv4 address of the machine on which the feature will be executed.
- `{{ local_admin }}`: Username of the global user.
- `{{ password }}`: Password of the global user.
- `{{ gateway }}`: IPv4 address of the default router's LAN side.

Adding to DB

Now that a feature has been created, the builder needs knowledge about its existence by inserting its location and its variables in the database. Following steps are required:

- Place the playbook file in `src/features`.
- Create an entry in the `features` table.
  - `name` is free of choice.
  - `description` is free of choice.
  - `path` is the relative path starting from `src` (E.g., `src/features/feature.yaml`)
  - `category` should match with the category of the templates with which the feature is compatible (`windows` or `linux`).
- Create for each non reserved variable an entry in the `options` table
  - `featureId` is the id given to the feature entry from the previous step.
  - `name` is the same name as the variable name used in the playbook (underscores included).
  - `type` is the input type expected (`string` or `file`).

Annex 11: Ansible Playbook Example: Promote to DC

```
---
- name: Create new Active-Directory Domain & Forest
  hosts: localhost
  connection: local
  gather_facts: false

  tasks:
  - name: Add host to Ansible inventory
    add_host:
      groups: temp
      name: '{{ address }}'
      ansible_user: '{{ local_admin }}'
      ansible_password: '{{ password }}'
      ansible_connection: winrm
      ansible_winrm_transport: ntlm
      ansible_winrm_server_cert_validation: ignore
      ansible_winrm_port: 5985

- hosts: temp
  gather_facts: false

  tasks:
  - name: Wait for system to become reachable over WinRM
    wait_for_connection:
      timeout: 900

  - name: Set static IP address
    win_shell: '(new-netipaddress -InterfaceAlias "Ethernet0 2" -IPAddress {{
address }} -prefixlength 24 -defaultgateway {{ gateway }})'
    async: 5
    poll: 0
    ignore_errors: true

  - name: Wait for system to become reachable over WinRM
    wait_for_connection:
      timeout: 900

  - name: Set upstream DNS server
    win_dns_client:
      adapter_names: '*'
      ipv4_addresses:
      - '{{ gateway }}'

  - name: Install Active Directory
    win_feature: >
        name=AD-Domain-Services
        include_management_tools=yes
        include_sub_features=yes
        state=present
    register: result
  - name: Create Domain
    win_domain: >
        dns_domain_name='{{ domain_name }}'
        safe_mode_password='{{ password }}'
    register: ad

  - name: reboot server
    win_reboot:
    when: ad.changed

  - name: Set internal DNS server
    win_dns_client:
      adapter_names: '*'
      ipv4_addresses:
      - '127.0.0.1'
```

Annex 12: API Endpoints

This is a list of all API endpoints. Body schemes can be found in annex 13 and 14.

CRUD operations

- `GET /scenarios`: Get all scenarios.
- `GET /scenarios/{name}`: Get all versions of a scenario.
- `POST /scenario`: Create new scenario. Body scheme in annex 13.
- `PUT /scenario`: Edit scenario. Body scheme in annex 14.
- `GET /scenario/{id}`: Get details of scenario.
- `DELETE /scenario/{id}`: Delete scenario.

Actions

- `POST /scenario/{id}/build`: Build scenario.
- `DELETE /scenario/{id}/destroy`: Destroy scenario.
- `PATCH /scenario/{id}/run`: Start scenario (turns all VMs on).
- `PATCH /scenario/{id}/stop`: Stop scenario (turns all VMs off).

Components

- `GET /templates`: Get all vCenter templates.
- `GET /features`: Get all features and their variables.
- `GET /subnets`: Get all used subnets.

Annex 13: Body Scheme to Create a Scenario

Following body scheme is used in the POST request to create a new scenario. The content type is `multipart/form-data` which exists out of `files` and `request` where `files` is a binary string and `request` of type `application/json`. The `request` looks like the following:

```
{
   "name": "string",
   "version": "string",
   "description": "string",
   "subnets": [
     "string"
   ],
   "machines": [
     {
       "name": "string",
       "hostname": "string",
       "cpus": 0,
       "memory": 0,
       "adminPassword": "string",
       "ipv4": "string",
       "template": "string",
       "features": [
         {
           "name": "string",
           "order": 0,
           "options": [
             {
               "name": "string",
               "value": "string"
             }
           ]
         }
       ]
     }
   ]
 }
```

- `name`: Name of the scenario.
- `version`: Version of the scenario (commonly in format of 1.0.1).
- `description`: Description of the scenario.
- `subnets`: A list of maximum two unique strings in 10.x.x.0 format or an empty list to get a subnet assigned by the builder. In case of an empty list, `machine.ipv4` must be `null`. Subnets have as subnet mask of 255.255.255.0. Find already used subnets with `GET /subnets`.
- `machine.name`: Name of the VM.
- `machine.hostname`: Hostname of machine.
- `machine.cpus`: Number of CPUs to be assigned to this machine.

- `machine.memory`: Amount of memory to be assigned to this machine in MB.
- `machine.adminPassword`: Administrator password of the machine (`null` for Linux machines).
- `machine.ipv4`: Address to be assigned to the machine. This can be an IPv4 address within the range of one of the subnets defined in `subnets` to assign the machine a static IP (e.g., `10.0.35.9`), or this value can be equal to one of the subnets to get an IP via DHCP ().
- `machine.template`: Template on which the machine is based. Find all templates with GET /templates.
- `machine.feature.name`: Name of the feature to be added to machine. Find all features with GET /features.
- `machine.feature.order`: Order in which the feature should be executed.
- `machine.feature.option.name`: Name of the option that is about to be assigned a value.
- `machine.feature.option.value`: Value of the option.

Annex 14: Body Scheme to Edit a Scenario

Following body scheme is used in the POST request to edit an existing scenario. The content type is `application/json` which looks like the following:

```
{
  "id": 0,
  "name": "string",
  "version": "string",
  "description": "string",
  "subnets": [
    {
      "id": 0,
      "subnet": "string"
    }
  ],
  "machines": [
    {
      "id": 0,
      "name": "router",
      "hostname": null,
      "cpus": null,
      "memory": null,
      "adminPassword": null,
      "ipv4": null,
      "template": "opnsense",
      "features": []
    },
    {
      "id": 0,
      "name": "string",
      "hostname": "string",
      "cpus": 0,
      "memory": 0,
      "adminPassword": "string",
      "ipv4": "string",
      "template": "string",
      "features": [
        {
          "name": "string",
          "orders": [
            {
              "id": 0,
              "order": 0
            }
          ],
          "options": [
            {
              "name": "string",
              "values": [
                {
                  "id": 0,
                  "value": "string"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

The scheme to edit an existing scenario is similar to the one to create a new scenario with some deviations:

All entries that have been created in the database when initially creating a scenario have gotten an id. When sending the request to edit a scenario, the builder will check the specified ids and edit the matching entries. That is the reason there are more objects to be found in this scheme. The needed ids can be found by issuing `GET /scenario/{id}`.

An `option` or `feature` does not require an id as these are static components.

Note that string `value` has become a list of values and string `order` has become a list of orders. The reason for this is the way that the database works. There is a table `values` which keeps track of the values that have been defined for an option of a feature for a machine. In other words, a value is bound to one option and one machine, but one option can have multiple values for multiple machines. That is why an option object has a list of values which will always contain only one value: The value that has a relation to both that option and the machine. Values that have a relation with other machines are filtered out.

In this scheme, the router machine is added to emphasise that it will be present in every scenario and must not be edited.

Annex 15: Test Scenario

```
{
    "name": "simple AD environment",
    "version": "1.0.1",
    "description": "Windows 2022 DC with joined Windows 10 and apache web server",
    "subnets": [
      "10.0.35.0"
    ],
    "machines": [
        {
            "name": "dc",
            "hostname": "KING",
            "cpus": 2,
            "memory": 4096,
            "adminPassword": "NVISO123",
            "ipv4": "10.0.35.254",
            "template": "win_2022",
            "features": [
                {
                    "name": "promote-to-dc",
                    "order": 1,
                    "options": [
                        {
                            "name": "domain_name",
                            "value": "test-domain.de"
                        }
                    ]
                }
            ]
        },
        {
            "name": "workstation",
            "hostname": "SQUIRE",
            "cpus": 2,
            "memory": 4096,
            "adminPassword": "NVISO123",
            "ipv4": "10.0.35.0",
            "template": "win_10_20H2",
            "features": [
                {
                    "name": "join-domain",
                    "order": 2,
                    "options": [
                        {
                            "name": "dc_address",
                            "value": "value": "10.0.35.254"
                        },
                        {
                            "name": "domain_name",
                            "value": "test-domain.de"
                        }
                    ]
                },
                {
                    "name": "disable-firewall",
                    "order": 3,
                    "options": []
                }
            ]
        },
        {
            "name": "debian",
            "hostname": "LACKEY",
            "cpus": 1,
            "memory": 1024,
            "adminPassword": null,
            "ipv4": "10.0.35.0",
            "template": "debian_11",
            "features": [
                {
                    "name": "apache-web-server",
                    "order": 4,
                    "options": []
                }
            ]
        }
    ]
}
```