

Analysis of Scalability in Declarative Deployment Environments

eine Fallstudie bezüglich der AIT Cyber Range

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Richard Stöckl, B.Sc.

Matrikelnummer 11908080

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Mitwirkung: Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Dr.rer.soc.oec. Florian Skopik

Tobias Pfaller, B.Sc. M.Sc.

Lenhard Reuter, B.Sc. M.Sc.

Wien, 12. November 2025

Richard Stöckl

Edgar Weippl



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Analysis of Scalability in Declarative Deployment Environments

in the Case of the AIT Cyber Range

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Richard Stöckl, B.Sc.

Registration Number 11908080

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Assistance: Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Dr.rer.soc.oec. Florian Skopik

Tobias Pfaller, B.Sc. M.Sc.

Lenhard Reuter, B.Sc. M.Sc.

Vienna, November 12, 2025

Richard Stöckl

Edgar Weippl



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Richard Stöckl, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 12. November 2025

Richard Stöckl



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

*Also ich glaube an Zufälle. Denn wissen Sie, Zufälle passieren jeden Tag.
Aber ich traue Zufällen nicht.*

— Elim Garak

Ich möchte mich hiermit bei all jenen bedanken, die mir ein Voranschreiten bis zu diesem Punkt ermöglicht haben. Allen voran meinem Betreuer Edgar Weippl und meiner Betreuung vom AIT, namentlich Florian Skopik, Tobias Pfaller und Lenhard Reuter. Eine intensive Betreuung in diesem Ausmaß ist ein Privileg. Insbesondere möchte ich mich bei Dennis Toth, Georg Reckendorfer, und Andrea, Luisa, Reinhard, Sabine, Kurt, Georg, Karl und Camill Stöckl bedanken.

Ich bitte hiermit um Verzeihung all jene Personen und Organisation, welche in dieser Sektion keine explizite Erwähnung gefunden haben. Ohne auch nur eine der Genannten wäre das in dieser Form nicht möglich gewesen.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I believe in coincidences. Coincidences happen every day. But I don't trust coincidences.

— Elim Garak

I would like to thank all those who have enabled me to progress to this point. First, my supervisor Edgar Weippl and my supervision of the AIT, namely Florian Skopik, Tobias Pfaller and Lenhard Reuter. Such an intensive supervision is a privilege. In particular, I would like to thank Dennis Toth, Georg Reckendorfer, and Andrea, Luisa, Reinhard, Sabine, Kurt, Georg, Karl and Camill Stöckl.

I apologize to all those persons and organizations who have not been explicitly mentioned in this section. Without even one of them, this would not have been possible in this form.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Cyber Ranges sind virtuelle IT/OT-Infrastrukturen, die eine sichere, aber dennoch realistische, Umgebung darstellen. Dadurch können zum Beispiel Cybervorfälle in einem realistischen Umfeld simuliert werden, um einerseits technische Fähigkeiten zu verbessern, andererseits Entscheidungs- und Kommunikationsprozesse zu testen und zu üben. Cyber Range Infrastrukturen haben hohe Anforderungen im Bereich Dynamik und Skalierbarkeit um regelmässig an unterschiedliche Trainingszenarien angepasst werden zu können. Mithilfe von Benchmarks können Defizite hinsichtlich der Skalierbarkeit in Cyber Ranges festgestellt werden. Diese Diplomarbeit mit dem Titel *Analysis of Scalability in Declarative Deployment Environments: in the Case of the AIT Cyber Range* beschäftigt sich vordergründig mit architektonischen Mustern zur Lösung der Herausforderung der Skalierbarkeit und wendet das Vorgehensmodell beispielhaft in der AIT Cyber Range an. Hierzu ist auf Basis der *Design Science* eine Methode konstruiert worden, welche anhand von definierten *Kick-out Factors (KOFs)*, architektonische Muster in Relevanzklassen für Cyber Ranges einteilt. Anschliessend sind Metriken definiert worden, anhand deren Experten für Cyber Ranges eine Rangordnung erstellt haben, um die architektonischen Patterns nach potentiellen Mehrwert für Skalierbarkeit in Cyber Ranges zu sortieren.

Zur Evaluierung von sowohl der Methode, als auch den Mustern an sich, sind die folgenden anschließend in der AIT Cyber Range implementiert worden: *Proxy Jump Component*, *Package Cache Component* und *Multi-Client Component*, welche leichte Abwandlungen von bereits aus der Literatur bekannten Muster darstellen. Mithilfe von Benchmarks ist dann eine Analyse durchgeführt worden, welche beurteilt welche Muster welchen quantitativen Mehrwert gebracht haben und ob sich die Vorhersagen anhand der Methode bestätigen. Vor allem das *Package Cache Component*-Muster hat mit seiner signifikanten Steigerung der Skalierbarkeit besonders hervorgestochen. Abschliessend werden qualitative Merkmale der Muster diskutiert, um mögliche positive und negative Auswirkungen aufzudecken, wobei hier das *Proxy Jump Component*-Muster erwähnenswert ist, welches es ermöglicht mehr Teilnehmer zu bedienen als bisher.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Cyber ranges are virtual IT/OT infrastructures that provide a secure nevertheless realistic environment. This allows cyber incidents to be simulated in a realistic environment, in order to improve technical skills on the one hand and to test and practice decision-making and communication processes on the other. Cyber range infrastructures have high requirements in terms of dynamics and scalability in order to be adapted regularly to different training scenarios. Benchmarks can be used to identify deficits in terms of scalability in cyber ranges. This thesis, entitled *Analysis of Scalability in Declarative Deployment Environments: in the Case of the AIT Cyber Range*, primarily deals with architectural patterns for solving the challenge of scalability and applies the process model to the AIT Cyber Range as an example. To this end, a method has been constructed on the basis of *Design Science* which uses defined *Kick-out Factors (KOFs)* to classify architectural patterns into relevance classes for cyber ranges.

Subsequently, metrics were defined, which experts for cyber ranges used to create a ranking in order to sort the architectural patterns according to their potential added value for scalability in cyber ranges. The evaluation of both the method and the patterns themselves resulted in the following being implemented in the AIT Cyber Range: *Proxy Jump Component*, *Package Cache Component* and *Multi-Client Component*, which are slight modifications of patterns already known from the literature. With the help of benchmarks, an analysis was then carried out to assess which patterns brought which quantitative added value and whether the predictions based on the method were confirmed. The *Package Cache Component* in particular stood out with its significant increase in scalability. Finally, qualitative characteristics of the patterns are discussed in order to uncover possible positive and negative effects, whereby the *Proxy Jump Component* is worth mentioning here, as it enables more participants to be served than before.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Motivation & Problem Statement	1
1.2 Contributions	2
1.3 Research Questions	3
1.4 Methodological Approach	3
1.5 Structure of the Thesis	4
2 Background & Related Work	7
2.1 Cyber Range	7
2.2 Virtualization	13
2.3 Infrastructure as a Service	15
2.4 Infrastructure as Code	16
2.5 Scalability	19
2.6 Strategy Patterns	20
2.7 AIT Cyber Range	20
3 Methodology	21
3.1 Problem Identification	21
3.2 Accomplishing the Objectives	22
3.3 Design and Development	25
3.4 Demonstration	25
3.5 Evaluation	26
3.6 Communication	31
4 Current State	33
4.1 Current Deployment Time in the AIT Cyber Range	33
4.2 Takeaways	40
	xv

5	Candidate Pattern Selection	43
5.1	Architectural Requirements of a Cyber Range	43
5.2	Finding Patterns from the Literature	44
5.3	Pattern Refinement	45
5.4	Pattern Ranking	48
5.5	Final Pattern Selection	52
6	Implementation of Candidate Patterns	53
6.1	Implementation Decisions of Candidate Patterns	53
6.2	Proxy Jump Component	53
6.3	Package Cache Component	56
6.4	Multi-Client Component	58
6.5	Summary of Findings	60
7	Discussion	61
7.1	Proxy Jump Component	61
7.2	Package Cache Component	62
7.3	Multi-Client Component	63
7.4	Pattern Combination	65
7.5	Implementation Effort	65
7.6	Research Questions	67
8	Conclusion	69
8.1	Future Work	69
A	Benchmarks	71
A.1	Proxy Jump Component	71
A.2	Package Cache Component	73
A.3	Multi-Client Component	78
	List of Figures	83
	List of Tables	85
	Acronyms	87
	Bibliography	89

CHAPTER 1

Introduction

This chapter outlines the motivation and problem statement of the study. It identifies existing research gaps and defines the intended contribution, which forms the basis for the research questions. The methodological approach used to address these questions ensures that the results are comprehensible. Finally, the chapter presents the structure of the thesis to help guide the reader.

1.1 Motivation & Problem Statement

Cyber ranges are virtual environments which allow performing exercises and trainings for students and professionals in an environment that is deployed separately from those used in production systems. The primary educational goal is to train skills in intrusion detection, digital forensics, log-analysis, security best-practices and similar in enterprise infrastructure. Other goals are training incident response processes, communication structures and reporting channels according to guidelines such as the Network and Information Systems (NIS) Directive [60] or the Digital Operational Resilience Act (DORA) [68] and the communication skills between different departments such as the management and technical departments [84, pp. 2, 3].

Since attacks on networks have become more frequent in recent times, cyber ranges have become increasingly relevant [52, pp. 1161, 1164]. The isolation from the production environment ensures that any potentially hazardous scenario can be conducted without any adverse impact on the infrastructure [55, p. 2].

Cyber ranges provide scenarios with numerous and diverse services to mimic real-world environments. This is accomplished through the utilization of specialized tools for deployment automation. This ensures that possible side-effects from diverting configuration in stand-alone deployments are covered. An added advantage of deploying the maximum number of services that the infrastructure and development resources can support

is that scenario testing can be conducted without necessitating the dismantling and reconfiguration of the environment.

However, with growing amounts of users and services to deploy, scalability issues occur, which will result in a significant increase in the time required to set up an exercise environment [64, p. 214, 80, p. 112, 47, p. 281]. There are two general stages of the deployment: the infrastructure provisioning and the software deployment. The scope of this work is limited to the software deployment, since it scales much worse than the infrastructure provisioning. To enumerate some of the software that necessitates deployment and configuration, namely the creation of user accounts, the generation of SSL certificates, the generation of SSH keys, the configuration of firewalls, and the installation of software services. This part is much more complicated and scales badly currently. Depending on the scenario, this process can take anywhere from a few hours to a few days. Furthermore, with an increasing number of services being deployed, the probability for errors increases and thus the number of services to recover as well. This also increases the necessity to appose scalability issues.

In contrast to production environments, exercise environments are frequently deployed and destroyed, thereby elevating the significance of the deployment duration. Furthermore, a major difference between production environments and exercise environments is that services and clients are deployed all at once, whereas in production environments this happens more widely distributed over time. Cyber ranges frequently employ Infrastructure as Code (IaC) due to the reduced effort required during deployments, which is a result of the frequent deployments.

Despite the existence of methods to improve scalability, there is currently no systematic overview that specifies the most effective and beneficial approaches in cyber ranges. The absence of such an overview renders the process of enhancing the scalability of cyber ranges both laborious and investigative.

Furthermore, there is no scoring or comparison scheme available which can be utilized to establish such an overview. Neither a comparison of the patterns nor a rapid evaluation of scalability improvement methods is possible in the absence of such a scheme.

1.2 Contributions

This work investigates to what extent architectural patterns improve the scalability of cyber range deployments. Pattern-based reasoning supports generalizable results across platforms. We implement and benchmark selected patterns to quantify their effect on deployment time.

A literature review resulted in the initial selection of architectural patterns. This necessitates the establishment of a process for scoring and comparing the numerous patterns available prior to their actual implementation. This process enables the analysis and systematic comparison of patterns using a designed scheme and predetermined

metrics in order to create a selection of impactful patterns that will be evaluated within an implementation.

The work encompasses not only the process itself but also the results of the pattern comparison in the context of cyber ranges. This illustrates how and how well the process of the pattern selection works using specific patterns.

The AIT Cyber Range serves as an evaluation platform for the practical implementation of the selected patterns. The evaluation results serve as benchmarks for the scalability of the patterns and as a qualitative discussion identifying potential implementation issues.

1.3 Research Questions

The aimed contributions of section 1.2 lead to the following research questions that will be answered in order to provide evidence for the findings:

RQ₁: To what extent can Pattern-based Deployment Models improve the scalability of Infrastructure as Code driven configuration architectures?

The benchmarks described in section 3.5 will be the primary source of information for addressing this inquiry. The outcomes will show changes with respect to their complexity class in addition to offering a comparison of deployment times in absolute terms.

RQ₂: Which metrics can be used to perform a ranking on patterns used in deployment models to perform a pattern selection process?

The solution to this inquiry is contained in section 3.2.3. The metrics are not only beneficial for this thesis, but they may also be employed in future research on comparable subjects, thereby ensuring that the results of this work are comparable.

RQ₃: Which measurements can be taken to improve the scalability of Infrastructure as Code driven configuration architectures beside pattern-based Deployment Models?

In addition to the streamlined methodology of this thesis, which involves the implementation of patterns following a selection process, improvements may arise during the implementation process discussed in section 3.4 that are not directly addressed by the pattern.

1.4 Methodological Approach

The method is based on the *Design Science* approach according to Wieringa et.al [90] and is structured as below in order to answer the research question:

1. **Identify Problem and Motivate.**

Identify the scalability issues that exist in cyber range deployments.

2. Define Objectives of a Solution.

a) **Literature Review.**

Identify potential patterns which can be used in order to optimize the performance and scalability of cyber ranges.

b) **Pattern Classification.**

Classify found patterns into relevance-classes to perform a pre-selection to reduce the number of suitable candidate patterns.

c) **Pattern Selection.**

Define metrics to estimate potential enhancements achieved by patterns. Subsequently, rank the pre-selected patterns according to the metrics and identify the most encouraging patterns.

3. Design and Development.

Implement the selected patterns using the AIT Cyber Range as an evaluation-platform. This agile process aims to improve the pattern in an iterative way.

4. Demonstration.

Present considerations, obstacles and potential issues of the implemented patterns. This knowledge is important when applying the patterns to other cyber ranges.

5. Evaluation.

Measure the improvement achieved by the applied patterns with different input sizes by comparing them to the benchmarks made during the problem identification.

1.5 Structure of the Thesis

Chapter 2 *Background & Related Work* introduces the cyber range domain. It establishes the lifecycle of a cyber range, including its security, availability, and compute resource requirements. In addition, it outlines the steps involved in the deployment of cyber ranges as well as the kinds of technology stacks that can be used.

In chapter 3 *Methodology*, a method is elaborated to find the most promising patterns for cyber ranges. To achieve this, the method of identifying patterns in the literature, performing a pre-selection, and organizing them to select the most beneficial is delineated. It also explains how the evaluation platform is constructed and how the outcomes of pattern implementations can be assessed.

chapter 4 *Current State* presents the baseline of the AIT Cyber Range. It describes the scalability of the deployment and established the foundation for selecting patterns and defining evaluation metrics.

A literature review is conducted in chapter 5 *Candidate Pattern Selection* to identify suitable patterns. Subsequently, Kick-out Factors (KOFs) and metrics are established to

either dismiss or advance patterns through the process and to allow experts to organize the patterns according to their anticipated potential in cyber ranges. Following the pre-selection and ranking of the experts, an ordering is established to ascertain which patterns to assess in the subsequent chapter.

The evaluation of the patterns selected in the previous chapter is accomplished by chapter 6 *Implementation of Candidate Patterns*. It is organized by pattern, delineates the implementation decisions, and illustrates the benchmarks as a diagram. Furthermore, highlights the key findings from each benchmark.

The outcome of the preceding chapter is examined in chapter 7 *Discussion*. Aspects of the results that are not directly related to benchmark results are discussed as well. This includes side effects, concerns, the implementation effort, and the significance of the methodology.

Chapter 8 *Conclusion* provides a summary of this work and potential connections that future research can build upon.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Background & Related Work

The purpose of this chapter is to explain the concept of a cyber range, identify the problems it addresses, review existing solutions, and outline how such an infrastructure can be built. The planning process consists of where the components are running on, how the platform is built, and how the components are provisioned and configured. After planning, a major issue – the scalability problem – is introduced, while also discovering the strategy patterns topic. This is discovered to help identify and solve these issues. Finally, the AIT Cyber Range will be introduced as an example of a cyber range that already exists in the industry.

2.1 Cyber Range

Cyber ranges are virtual exercise environments which allow performing trainings for students and professionals in an environment that is deployed separately from those used in production systems. They attempt to imitate the IT infrastructure of companies. These could be either near-complete replicas of real world infrastructures or idealized infrastructures. While replicating the infrastructure of companies has the benefit of being more realistic, it is much more work to setup and requires more resources and would not address the learning objectives any better nevertheless. Imaginary companies, however, are much less work to setup and have the benefit that they can be reused to a certain extent for different participating companies.

As seen in fig. 2.1, an example infrastructure may consist of three network-tiers: The Operational Technology (OT) (red), the Information Technology (IT) (blue) and the Demilitarized Zone (DMZ) (green). Each are separated by a firewall to ensure strict isolation. The OT-segment contains machines for physical operation such as the physical devices itself or the workstations the operators work on. In the IT-segment, regular clients, administration clients and internal services are located. The DMZ contains

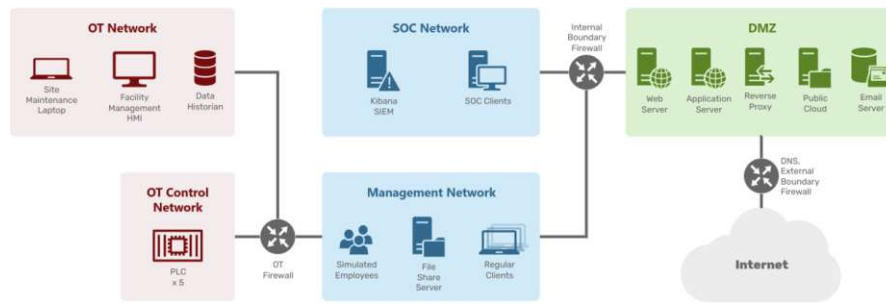


Figure 2.1: Example Infrastructure [69]

services which should also be reached externally from the internet such as web- or the mail-servers.

To realize such an infrastructure, a virtualization platform is required. This platform hosts all necessary virtual machines for servers, clients, firewalls and networks. The provisioning of that infrastructure is automated by infrastructure provisioning and configuration management tools. A more detailed elaboration can be found in section 2.1.2.

2.1.1 Cyber Range Exercises

Cyber range exercises can cover very different and a diverse use-cases.

Categories of cyber exercises are training of response cooperation, evaluation of cyber-security skills, information assessment and training real-world scenarios for skill-improvement technical exercises and organizational exercises according to Seker et.al. [74].

Cyber exercises may also follow a hybrid strategy where the above mentioned categories are applied to each company via roles. This approach allows that both, technical and management aspects can be trained at once. To implement this, each participant gets a role assigned which may be more on the technical side such as Computer Security Incident Response Team (CSIRT) or on the non-technical side such as management or public relations. The goal in such a hybrid scenario is that the role cooperate with each other in a such that they prevent cyber attacks, defend them if they happen and execute the incident response correctly. This enables that the participants learn technical, communication and organizational skills to successfully perform the incident response.

For organizational purposes, cyber exercises are separated into different teams. These teams are competing against each other or working together, depending on the scenario. Each team has different tasks and requirements in order to be able to participate in each. While this list is not complete, it enumerates the most important team colors according to Yamin et.al. [91, p. 11].

Red Team The read team is comprised of individuals who attempt to breach the infrastructure. They have to satisfy the tasks and boundaries predefined by the

white team. They are not allowed to attack the underlying infrastructure where the cyber range is deployed at, for instance.

Blue Team The blue team tries to defend the infrastructure actively against the red team. This encompasses tasks such as intrusion detection, detecting existing security vulnerabilities, and other related tasks. When detecting a cyber incident, they are responsible for starting the necessary processes according to the GDPR and NIS and communicating the incident within the team.

White Team The objective of the white team is to devise and execute scenarios for the cyber range. Furthermore, they are responsible for managing the exercise itself and giving other teams hints if they are stuck in progress.

Green Team The green team is responsible for ensuring the exercise's infrastructure. Before the actual game begins, they prepare the infrastructure. During game play, they monitor the deployment and fix failing components.

Defining what a cyber range contains and what it is about is not enough to fully grasp potential scalability issues. To make it more clear where certain issues may arise, the process of setting up a cyber range will be introduced in the next section 2.1.2.

2.1.2 Lifecycle

The establishment and execution of the method further described in chapter 3 necessitate clarification of the cyber range's lifecycle. In fig. 2.2, the lifecycle is illustrated in detail.

One high-level objective of a cyber exercise is the initial phase of the cyber range lifecycle, which entails the development of the exercise scope. The purpose of this is to determine the types of exercise-tasks and attacks that should be included in the exercise at a later time. It involves the preparation of concepts that comprise the necessary vulnerabilities and attacks, which will be implemented at a later time. This leads to a timetable that specifies the commencement and conclusion of each attack.

The tasks of the exercise for the participants to solve, can be concurrently planned, developed, and integrated following the planning phase.

Initial, the task must be selected from an existing catalog, novel vulnerabilities, or even from scratch. Considerations such as the complexity of exploit execution and defense are assessed in this context.

If it has not already been developed, this is completed during the subsequent phase. In an effort to expedite the development process, the task is tested outside the exercise environment. Testing the anticipated scenarios and vulnerabilities is the subsequent phase. It also encompasses the development and configuration of the attack vectors and scenarios.

Once the scenarios and attacks have been refined, they must be incorporated into the cyber range deployment plan. In the context of the AIT Cyber Range, this entails the

2. BACKGROUND & RELATED WORK

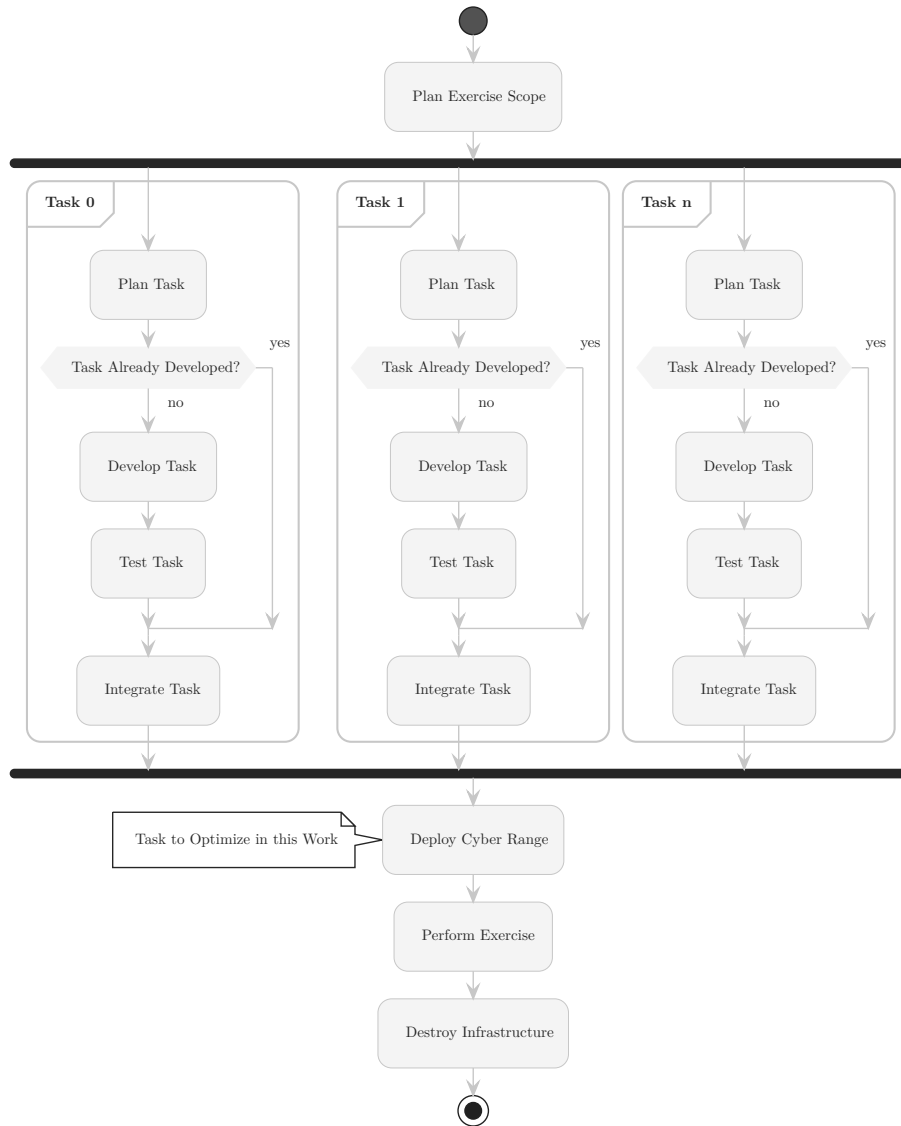


Figure 2.2: Cyber Range Exercise Lifecycle

creation of Terraform modules and Ansible playbooks for the IaC repository. This phase may involve the initial stages of minimal deployments to evaluate the newly integrated scenarios.

Afterward, the productive environment undergoes the full deployment of the cyber range. The capacity to work concurrently is significantly diminished at this juncture, as a cyber range exercise is a single deployment. The participants will subsequently utilize the productive environment in the exercise scenario. This is the first step that needs a full infrastructure for production. This is also the stage that this thesis tries to enhance.

The main stage phase – performing the exercise of the cyber range – commences following the deployment. During this phase, the participants commence the actual exercise by addressing, defending, and reporting the predetermined issues in the scenario. The maximum duration of this phase is a few days.

The infrastructure is destroyed in the final phase after the main phase concludes. This guarantees that future exercises can be conducted without incident. Additionally, it decreases the cost of compute resources or makes the infrastructure available for other workloads.

In contrast to other deployments that are intended to operate for years or even decades, a cyber range deployment that lasts only a few days is exceedingly brief. Moreover, the deployment time is not negligible in comparison to other deployments.

2.1.3 Security Considerations

The security requirements of a cyber range are distinct from those of typical software deployments for a variety of reasons. These requirements must be meticulously refined to enable the selection of patterns in a rational manner.

In a cyber range, the potential for environmental exploitation is a significant security concern. Users typically require the use of tools that are designed for conducting cyberattacks in order to engage in the exercise. Nevertheless, these instruments may also be exploited to inflict harm on the exercise environment. The result could be anything from the interruption of the exercise itself to the execution of actions with legal repercussions on the internet, as well as the reservation of new compute resources in the virtualization environment that could incur expense. This necessitates that participants are unable to exit their exercise environment to the greatest extent possible.

In certain specialized cyber exercises, it is feasible that the customer who orders a cyber range provides sensitive data to the exercise in order to enhance its realism. Such data must be handled with utmost care, with the potential to exclude specific components or patterns from being utilized in the scenario. For instance, the utilization of third-party services or cloud providers may be prohibited for such exercises, which consequently diminishes the potential strategy patterns that could be implemented in such circumstances.

However, there are also some security requirements that are more relaxed than those found in typical production environments. In productive environments, the security of internal services accessible to the participant against malicious behavior is of greater importance than in a cyber range. In reality, the purpose of a cyber range is to serve as a cyber security training environment, which means that participants are expected to conduct attacks on the services. This relaxation permits the utilization of patterns that are reusing cryptographically expensive artifacts, such as certificates, password hashes, or tokens.

In summary, the security requirements of a cyber range are neither strictly greater nor strictly less than those of a typical software deployment. This implies that there is no absolute limit that precludes specific patterns from being implemented in systems with “higher” security requirements, or vice versa.

2.1.4 Availability

In order to select suitable patterns, it is also necessary to consider availability as a qualitative criterion. The operation phase of a cyber range is typically much shorter than that of typical software deployments, as previously delineated in section 2.1.2. Consequently, the availability requirements are slightly different.

A cyber range is utilized by the customer for a brief period, in contrast to software services that are permanently consumed by customers over an extended period. The customer is not reliant on the cyber sector to maintain subjects such as productivity, security levels, or production lines.

Following the deployment during the operation phase, software and system maintenance is a significant factor that impacts availability. In order to prevent significant downtime or malfunctions, systems that are continuously operational for an extended period must be maintained in a variety of ways. Some of these maintenance actions include the monitoring and cleaning up of logs to prevent major outages and storage shortages, as well as the regular updating of software to prevent bugs that result in data loss or unresolved security vulnerabilities. Although these items should be kept in the deployment plan for cyber ranges, because of their brief lifespan, they are insignificant during the operational phase. This implies that patterns can be implemented in a cyber range that are in direct opposition to the maintenance requirements of conventional production environments.

2.1.5 Deployment Automation

For every exercise, the infrastructure has to be reset to its original state or prepared for a different exercise. Deploying lots of infrastructure manually takes a lot of time and is very error-prone. Therefore, it is desired to automate this process as much as possible to reduce the workload of green teams [53, p. 8].

The automation of exercise deployment can be divided into two distinct layers. The bottom layer is responsible for requesting the necessary resources, such as virtual machines and virtual network infrastructure. Additionally, there is a layer responsible for configuring all the software components running on the servers and the clients. Although they share similarities, they have to be handled separately [54, p. 3].

Even with automation, the deployment of an exercise can consume hours to nearly a day. Not only that, this means that the deployment must be started long before the beginning of the exercise. It also implies that the development and testing of exercises consumes a considerable amount of time, and in the event of any errors during deployment, they must be rectified and re-deployed, which also entails a significant amount of time.

To automate a deployment, certain requirements have to be met to make this possible. These are virtualization, a platform for orchestration such as Infrastructure as a Service (IaaS) and tools which perform the deployment such as Infrastructure as Code (IaC).

2.2 Virtualization

Virtualization allows the deployment of components that are normally physical devices onto a software stack instead. This enhances the flexibility of device provisioning. Proper solutions allow for the moving of virtual resources to other machines where the software virtualization stack is installed. Therefore, virtualization improves the availability of resources while also being more flexible in how resources are allocated, which allows available hardware to be better utilized. The software that is doing this is called a *hypervisor* [30, pp. 1, 2].

Another benefit of virtualized resources over physical resources is that their provisioning can often be done using an API. This not only facilitates the provisioning of new systems by software, but also expedites the process. It is therefore much faster to provision hundreds of virtual resources than to provision physical systems [53, p. 8].

2.2.1 Virtual Machines

Virtual machines are the partitioned resources defined in virtualization. The partitioned resources can be viewed as virtual computers that operate on a virtualization host. The use of virtual machines requires that the host split its resources, such as the CPU, main memory, storage, and GPU, into multiple parts. The virtual machines therefore are able to access a portion of the hosts resources. This also reduces the number of physical machines required [30, pp. 1, 2].

There are multiple options available for resource allocation. One strategy is that the host allocates a predetermined quantity of resources to the virtual machines. This approach guarantees that a virtual machine will have access to all resources dedicated to it at any time, but it reduces flexibility and raises the issue that if another virtual machine runs out of resources, it will not have access to further resources even if the other virtual

machines do not fully utilize the allocated resources. To prevent this, resources can also be allocated on demand according to the current requirements of the virtual machine [63, pp. 2, 3].

Virtual machines, similar to physical machines, necessitate access to other machines and network resources. Network topologies are typically realized via *Software Defined Networks (SDN)* [48, p. 12].

2.2.2 Virtualization Approaches

Virtual machines are a concept that describes how virtual computers are isolated from each other and the host computer. However, virtualization it is a generalized technique, which can be implemented in different ways with emphasis on different aspects such as performance or compatibility. According to Armstrong and Djemame [8, p. 5], there are three different major types of virtualization:

Full Virtualization All the resources of the guest can be virtualized using hardware features from the host in this type of virtualization. Each component to virtualize, such as the CPU, main memory, storage and other devices, will have a virtual device created. Normal CPU instructions are directly executed on the host's CPU. When a privileged instruction is encountered, the hypervisor attempts to emulate it.

A major constraint of full virtualization is that it is required that the guest runs on the same CPU instruction set.

Paravirtualization Paravirtualization enables the virtualization guest to access the host system's API calls with specified *hypercalls*. The benefit of this is that the guest can do system calls directly on the host instead of having to isolate and interpret them.

To use virtualization, the guest system must be ported to use it. Since the API is hypervisor-specific, it must be done for every host system the guest system should run on. This is not always feasible on closed systems as they are not easily modified. Therefore, often only device drivers for hardware such as storage, network, or graphics are implemented. These drivers are comprised of two distinct components: a frontend driver for the guest and a backend driver that implements the hypercalls.

Most prominent hypervisors include KVM and XEN.

Hardware-Assisted Virtualization This type of virtualization requires the host CPU to implement special instructions in order to achieve virtualization capabilities. This eliminates the need to implement dedicated drivers. Thus, hardware-assisted virtualization is the best performing of the three. Additionally, the guest does not have to implement guest specifics such as hypercalls.

The major drawback is that the guest has to be running on the same instruction set as the host and the host has to provide special instructions.

Virtual machines themselves only define how resources of a physical machine can be partitioned, but not how the underlying infrastructure looks. In order to resolve that, virtualization platforms are required to manage virtual machines. Virtualization platforms provide mechanisms to manage the life-cycle of virtual machines and provide accountability to ensure proper cost calculation. Two prominent examples are OpenStack and libvirt.

2.3 Infrastructure as a Service

Infrastructure as a Service is a type of virtualization platform that offers users to book virtual infrastructure on demand. Part of this infrastructure are typical virtualized hardware such as CPU power, main memory, storage, GPUs or network resources.

IaaS providers commonly expose APIs that enable programmatic provisioning, drastically reducing manual effort [75, p. 2].

2.3.1 Cost

The IaaS model relies on a pay as you go model. This implies that a user solely pays for the resources they are currently utilizing, typically with a precision of minutes. In certain instances, it is distinguished whether a resource is active or inactive, wherein inactive resources incur a lower cost. For instance, a compute node may be operating in standby mode. Since it does not require any main memory or CPU at this time, it can be less expensive. However, the storage cost of this node is still non-zero.

In contrast to on-premise solutions, IaaS does not have upfront costs. It is not necessary to purchase expensive hardware upfront or to plan and execute complex setups in both the hardware and software aspects.

2.3.2 Availability

Typical setups of IaaS backends involve computation clusters. They are responsible for scheduling all resources across the cluster nodes properly. This ensures that instances of provisioned infrastructure can be run in a high availability mode with near to zero downtime [75, p. 2].

The degree of availability of such infrastructures is usually part of a Service Level Agreement (SLA). Depending on how much a user is willing to pay, the percentage of the guaranteed availability might be lower or higher [75, p. 2].

Cloud service providers typically offer different regions in which the virtual infrastructure will be located. The benefit of this is that, in case of an outage, only the resources of the particular region will fail. Drawbacks are latency and potential and higher cost in certain regions [75, pp. 2, 3].

IaaS is available from different vendors as different implementations. Vendors include Amazon Web Services [19], Google Cloud Platform [20], Microsoft Azure [21] and

OpenStack [61]. While the first three are closed source and are hosted by the vendors, OpenStack is open-source and allows for the hosting on premise.

2.3.3 OpenStack

OpenStack is a cloud computing platform which provides infrastructure as a service functionality. In that manner it shares many similarities with platforms from commercial vendors such as Amazon AWS, Google Cloud Platform, Microsoft Azure, et cetera. However, OpenStack is free and open-source meaning that in contrast to the other mentioned solutions above, OpenStack can be hosted on-premise. In addition, there are still vendors which provide IaaS with OpenStack which has the benefit for prohibiting vendor lock-in [12, p. 1].

If hosted on-premise, OpenStack inherits all benefits and drawbacks of these type of hosting such as privacy or service cost [12, p. 1]. OpenStack itself is not a single software artifact but rather consists of many components which are part of the OpenStack architecture. To name a few, there is *Nova* which is responsible to provision the virtual machines, *Neutron* which manages the network for the VMs or *Swift* which acts as the storage backbone [70, p. 1].

OpenStack itself does not include a hypervisor, meaning that the user is responsible for setting up the desired hypervisor [70, p. 2]. This has the benefit that the OpenStack infrastructure can be tailored for the specific use-case. Different hypervisors provide different abilities making it a critical decision regarding which CPU architectures are required and also the performance requirements.

OpenStack itself offers only the virtualization platform, but does not provision resources on its own. In order to perform the workloads as desired, the virtual machines must be created by calling OpenStack's API. This provisioning can be accomplished either manually or automatically.

2.4 Infrastructure as Code

Infrastructure as Code (IaC) specifies infrastructure in source code, improving reproducibility and integrating naturally with version control for collaborative change management.

2.4.1 Infrastructure Provisioning

Infrastructure Provisioning describes the process of allocating the virtual resources required for scenarios such as a cyber exercise. As infrastructures grow, provisioning complexity, and thus effort increases. In addition, provisioning must be reproducible to enable reuse of developed scenarios.

Manual provisioning does not scale well. Although writing automation scripts requires upfront effort, repeated deployments requires minimal additional cost. Automation reduces errors and improves speed through scripted, idempotent workflows [53, p. 1].

In order for an infrastructure deployment to be automated, it is imperative that the provider offers API support. In cloud-native scenarios, this is almost always the case, but it does introduce the difficulty of vendor lock-in since not every cloud service provider uses the same API. However, there are abstractions available to prevent this from happening [43, p. 5].

Programs that are used to perform automated deployments are known as *Infrastructure-as-Code* utilities. There are two major types of statements: declarative (specify the desired end state) and imperative (specify the steps to reach it). Declarative models evaluate current state and converge toward the target; imperative models execute commands in a fixed sequence. The declarative approach simply defined what had to be done, the imperative approach defined how it had to be done [53, p. 8]. Listing 2.1 shows an example on how Vim can be installed in an imperative way and in a declarative way using Ansible. While the first installs it regardless of the current state, the latter only installs it if necessary since its task is to bring the system in the described state instead of simply running commands.

```

1 - name: Ensure vim is installed (imperative)
2   ansible.builtin.shell:
3     cmd: apt install vim --yes
4
5
6 - name: Ensure vim is installed (declarative)
7   become: true
8   ansible.builtin.apt:
9     name:
10      - vim

```

Listing 2.1: Installing Vim imperative versus declarative with Ansible

Since Infrastructure as Code (IaC) files are text files similar to regular source code files, they inherit their advantages. One opportunity lies in using a version control system such as Git. This simplifies the process of how the infrastructure was defined in the past and makes it transparent on who changed something. Besides that, collaboration is much more convenient than in classic file shares due to the built-in automations for merge strategies when multiple authors change the same file [53, pp. 11, 12].

There are multiple tools available for infrastructure provisioning. Each of the closed-source vendors mentioned above offers its own provisioning tools [88, 33, 11] that are tied to their specific IaaS platform. Platform independent tools are Terraform [81], Pulumi [65] and OpenTofu [62]. Pulumi is not a full-blown tool for infrastructure provisioning itself but provides SDKs for various programming languages instead. Although this approach is quite flexible, it also means that a programming environment is necessary to provision infrastructure. Terraform and OpenTofu use the same domain-specific language

(HCL) since OpenTofu was a fork of Terraform when Terraform was open-source. Projects created before OpenTofu existed therefore tend to use still Terraform when trying to migrate to OpenTofu.

Terraform is a tool developed by HashiCorp for infrastructure deployment, with a primary focus on its ease of portability to specific cloud providers. The compute resources are defined using a declarative Domain Specific Language (DSL) – called HashiCorp Configuration Language (HCL) – to ensure human readability. In general, Terraform operates by creating a plan for infrastructure and dependencies that it must provision from the current HCL, and optionally, a state that was previously created by Terraform, ensuring that only compute resources that are not already existent on the provider are created. This plan can then be reviewed by the user and applied if everything goes as expected [43].

Infrastructure provisioning ensures that compute resources are available to the issuer. However, these resources are only provisioned, and they will not be able to handle the desired workload out of the box or work together. In order to overcome the limited configuration options through infrastructure provisioning, a method to do so is necessary.

2.4.2 Software Provisioning

When the infrastructure provisioning is finished and all components are reachable, the software has to be provisioned and configured to work properly. A major part of the software provisioning is to configure all components to work together. That is typically the case in a cyber range environment where services are configured to reach each other or where resources such as documents or client software are getting distributed as stated in [54, p. 4].

One significant advantage of automated configuration management is the reduction of configuration drift. The term *configuration drift* refers to the breach of configuration integrity when a similar or identical configuration is applied to multiple instances. Configuration drift typically occurs when a configuration option must be modified across multiple instances. In a manual process, the administrator might forget to apply the new configuration option on some instances [53, p. 8].

Similar to infrastructure provisioning, there are several tools available for software provisioning. Some of them are Ansible [6], Chef [18] and Puppet [66]. Although all of them provide at least an open-source version, Ansible is by far the most used by the industry, according to Guerriero et al. [34, p. 6].

Ansible [6] is a configuration management tool originally designed for Linux instances. The configuration process is solely performed on the client's end, requiring only the installation of Python on the host. Ansible automatically sends all the required modules to the desired host. The connection itself is maintained with SSH [59, p. 3].

Ansible is divided into three major subjects: tasks, roles, and plays.

Ansible tasks are atomic declarations that specify what needs to be configured. Examples include copying a file, editing a file, installing a package, etc. They are backed by modules which are implemented in Python which are either built into Ansible or provided by third-party sources [59, p. 3].

Ansible roles refer to a series of tasks. They group tasks together in order to provide a more abstract approach to arranging tasks. Typical roles include installing a web browser. In order to work, such a role must install packages, edit, and create files. In numerous instances, roles are parameterized by variables, enabling the caller to select configuration options such as the installation path, certain enabled or disabled features, and so forth [59, p. 3].

Ansible plays are even more abstract than roles. Roles are grouped together to form a coherent recipe. An example would be a client configuration playbook, which would include a web browser installation role, a desktop environment installation role, and an email client installation role [59, p. 3]. A collection of Ansible plays is called Ansible playbook.

Ansible employs inventories to register all of its hosts. In each playbook, it can be specified on which host or host groups they should be executed. This allows for fine granular control over which tasks to run on which hosts without copying and modifying playbooks or running them against different hosts manually [59, p. 3].

2.5 Scalability

Scalability in a cyber range is required in order to handle different manifestations of the concrete infrastructure. Depending on the infrastructure itself, the number of servers might vary significantly, meaning that more or less compute clients and more or less server resources might be required. Moreover, the size and complexity of the components of the infrastructure may vary depending on the concrete scenario to exercise. The issue here is that, regardless of the exercise size, the deployment time should be kept as short as possible.

Scalability is the ability of a system to respond to these changes in resources in a certain dimension. The dimension could encompass any combination of resources, including but not limited to CPU power, network throughput, multiple nodes, enhanced memory, and so forth. By enhancing the value of a particular dimension, a scalable system implies an enhancement in performance without altering the architecture itself. Scalable systems are crucial in situations where a task necessitates execution on a distributed system or when the available system resources undergo significant changes [15, p. 1].

2.6 Strategy Patterns

Software design patterns are a well-established approach to software development in the domain of software engineering. Their objective is to introduce patterns that can be applied to certain software architectures in order to reuse existing parts of successful architectures. Their benefit is to unify the design of software, which makes it easier for new developers to develop existing software, thus reducing the required training period [48].

Software design patterns were not formalized before they became used. Instead, they were created by experienced software engineers with expertise from already existing software, which led to an empirical approach [48].

2.7 AIT Cyber Range

The AIT Cyber Range [54] is an example of an implementation of a cyber range that already exists in the industry. As described in chapter 3, it is used as test, demonstration and evaluation platform in this work. From the bottom up, the AIT Cyber Range utilizes OpenStack as its virtualization platform due to its open source nature and high prevalence. The infrastructure provisioning is done with Terraform since it is the most widespread option of infrastructure provisioning tools according to Guerriero et al. [34, p. 4]. For software provisioning, Ansible is used for the same reasons.

Methodology

This chapter describes how the scientific method is constructed in order to ensure the reproducibility of this work. The method is based on the *Design Science* approach according to Wieringa et.al [90].

3.1 Problem Identification

Cyber ranges must be capable of being utilized in a variety of scenarios in a flexible manner. This is imperative to avoid the need to develop entirely new code for infrastructure for each scenario or exercise. A cyber range should be capable of adjusting the number of client machines that can be deployed for participant interaction. This is necessary to handle a greater number of participants than the scenario was originally designed for, as well as to conserve resources when they are not needed.

Empirical benchmarks show that the AIT Cyber Range requires a significantly longer deployment time when a greater number of clients are required. The AIT Cyber Range's current state is recorded throughout this work utilizing benchmarks, as time is the subject of optimization. The benchmarks can be employed to identify the particular problems regarding scalability and approximate the location of the scalability issue. These benchmarks rely on the following dimensions:

Clients Clients are the input size at which the AIT Cyber Range exhibits poor scalability characteristics. Figures show 0–40 clients in steps of five on the x-axis.

Playbooks For playbooks, different diagrams are used. Each playbook has an effect on various parts, so they each behave differently in terms of the number of clients. Some playbooks in the AIT Cyber Range are optional. To make the significance as general as possible, only the playbooks are evaluated which are required for every scenario. These are:

Bootstrap sets up proxies, disables firewalls temporarily and updates software. Affects all machines.

DNS installs and configures DNS services.

Client installs required client software and configures resources such as browser-bookmarks, email-accounts and password-manager databases. Affects client machines.

Learners builds, installs and configures the platform for participation access. Affects the machine used for the noVNC platform for installation and all other machines for gathering facts in order to configure learners – the noVNC platform of the AIT Cyber Range.

Certificates generates and installs certificates on servers and distributes them to all machines. Affects all machines.

User Management creates the required user accounts. Affects all machines.

Post resets configuration only required during the deployment process. Affects all machines.

Warm/Cold *Cold* runs execute immediately after infrastructure provisioning on clean hosts; *warm* runs repeat the same configuration on already configured hosts. This helps to identify which scalability issues related to the configuration workload and which are caused by the configuration management tools itself such as checks. Distinct diagrams are provided to represent this dimension.

To determine the quantity of the severity and show the problem, each of the playbooks was benchmarked. The exact structure of the resulting diagrams is elaborated in chapter 4.

3.2 Accomplishing the Objectives

This part shows how a literature review was used to find patterns. The patterns that were identified were subsequently categorized in order to refine them. The main selection process then ranks the patterns so that the best-performing ones can be used first.

3.2.1 Literature Review

The aim of the literature review in this particular work is to identify potential patterns and methods which can be used in order to optimize the performance and scalability of cyber ranges.

Scope of the Literature Review

This thesis's primary objective is to identify and categorize patterns that resolve scalability-related issues. Therefore, the initial step is to search the literature for existing patterns.

A significant component of this project is the utilization of benchmarks to investigate the impact of patterns on scalability. In order to guarantee the reproducibility of the benchmarks, it is crucial to explore the existing literature regarding infrastructure deployment benchmarking.

In numerous instances, patterns are not designed to improve the deployment time; rather, they are intended to combine architectures or increase modularity. Although these patterns are prevalent and beneficial in the field of software engineering, they are not included in the scope of this work because they do not have a direct correlation with deployment performance.

Literature Sources

Although there are numerous publication formats for scientific research, including written literature and oral transmission of knowledge, the majority of computer science-related knowledge is disseminated via the internet. Google Scholar remains one of the most widely used search engines for online literature. This thesis employs it not only to maintain its own search index but also to perform searches on other engines, including IEEE Explore and the ACM Digital Library.

Since it is evident that the literature already contains many cloud computing patterns, the following approach has been established to handle this information effectively and ensure its reproducibility.

3.2.2 Pattern Relevance Classification

As previously mentioned in the background chapter, architectural patterns are employed to identify potential scalability issues and identify methods to resolve them. Literature that is currently accessible serves as an asset for their identification. In order to reduce the large number of patterns, their relevance for cyber ranges is estimated. To determine whether a pattern is further examined or not, the score is divided into classes.

After the implementation of a preliminary framework for pattern classification, it is necessary to establish a procedure that exclusively examines patterns that have the potential to be beneficial to a cyber range.

3.2.3 Pattern Selection

From the literature review, it is evident that there are numerous patterns that are accessible. It is not feasible or advisable to implement all of them due to the amount of patterns already existing. Consequently, we developed a selection process to guarantee that the most promising patterns are implemented. The pattern selection is a method ensuring that of the proposed patterns found in the literature get filtered in way that if implemented, they will actually have the highest benefit on the scalability.

To establish the pattern selection process, a ranking will be determined by evaluating metrics that are discussed in section 5.4. Each pattern is rated by four experts in the field

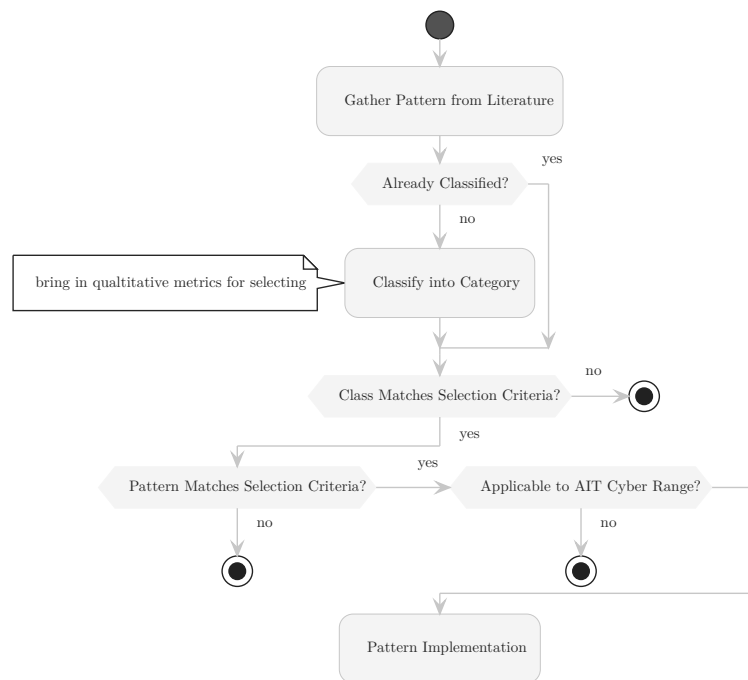


Figure 3.1: Pattern Selection Activity Diagram

of cyber ranges based on these metrics using estimates according to the *Delphi Method* [57]. This ranking is beneficial in identifying the potentially most useful patterns prior to their actual implementation, which facilitates the identification of which patterns to begin implementing and establishes a reasonable limit on the number of patterns that should be implemented.

The ranking itself which is further elaborated in section 5.4 can be divided into three distinct levels. The first group comprises the patterns that exhibit the most promising outcomes. Within the scope of this thesis, these patterns will be implemented. The patterns that are contained at the lowest level are not to be implemented at all, as they do not provide sufficient indication to warrant implementation. In between, there is the second level which contains the patterns that show some potential, but not a lot.

The resulting activity diagram of the literature review and pattern selection process is displayed in fig. 3.1.

Inductive Improvements

To initiate the implementation of concepts that are anticipated to necessitate a significant amount of effort, inductive improvements may be employed. The primary objective is to apply the pattern exclusively to a straightforward task. The implementation does not require production readiness. For instance, mocks may be implemented in place of genuine implementations of suspected performance bottlenecks to effectively evaluate the

concept.

An alternative strategy is to refrain from implementing it on the AIT Cyber Range. It is feasible to evaluate these patterns on a module from the Ansible Galaxy repository as an alternative. Despite the fact that this requires less effort than the full implementation for the AIT Cyber Range, the improvements can also be redirected to the Ansible Galaxy project. Moreover, the AIT Cyber Range will also benefit from the improvements if these roles are successfully implemented in projects that utilize the Ansible Galaxy repository.

The outcome still indicates whether there is an improvement, despite the fact that it is not yet ready for production, albeit with less significance. This approach enables the identification of potential patterns that may be worth further investigation in the future.

3.3 Design and Development

Throughout work, the design and development phase is an agile process. This results in an improvement in the efficiency of the implementation and evaluation process. As shown in fig. 3.2, the design and development part is executed as following:

1. Implementation of the selected pattern
2. Performance benchmarking and further analysis
3. Identification of possible improvements
4. Implementation of the identified improvements
5. If they actually improved the deployment-time, the process is done
6. Otherwise, repeat from step 3 onwards unless unsuccessful for two times

3.4 Demonstration

The proposed scalability changes are tested after the selection and evaluation of which patterns to implement to what extent. Consequently, the actual implementation must be completed. It is the objective of this section to offer a perspective on the potential issues, obstacles, and considerations that may arise during the implementation process. This thesis will incorporate the knowledge acquired through this process. In particular, it is possible that patterns were not sufficiently filtered during the selection process, resulting in non-negligible details that exacerbate the development effort or cause conflicts with other patterns. This approach enables the reader to make decisions regarding the pattern to be implemented based on a greater understanding of the subject matter than the selection process itself.

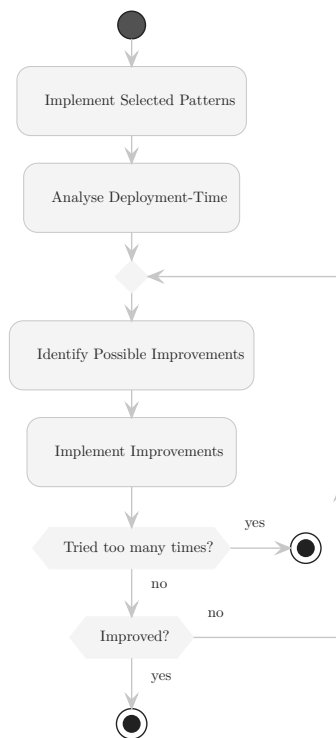


Figure 3.2: Design and Development Activity Diagram

3.5 Evaluation

The scalability issues are outlined and ranked in a manner that is clear, allowing for the identification of the areas that require attention during the eradication process. In order to verify and demonstrate the actual improvement, the benchmark is repeated following the completion of the implementation. The benchmarking methodologies are predominantly implemented in accordance with the recommendations of Bartz-Beielstein et al. [13].

3.5.1 Metrics & Key Performance Indicators

Key Performance Indicator (KPI) are the metrics of a benchmark which define the attribute to be measured and therefore to be improved.

There are many metrics available for benchmarking software deployments:

Latency is the time difference between starting to request a certain task and when it actually starts.

Throughput is a metric that measures how many tasks can be handled within a time frame.

CPU Usage how much of the CPU is consumed. In Ansible, certain tasks require more CPU power than others. Examples of tasks that consume more CPU resources are certificate generation, archive compression/decompression, and hashing.

Memory Consumption defines how much memory the process consumes. When the memory is near capacity, the throughput of tasks cannot be increased anymore.

Network Throughput is influenced by how much data is sent through the network. When sending big datasets to instances installing packages on many hosts in the network, this can be effected negatively.

Disk I/O is important when sending big data sets to hosts or installing new software.

Deployment Time is the KPI which is the result of all above metrics.

Within the context of a cyber range, the deployment time is the primary KPI. It represents the time difference between the commencement of a scenario deployment and its conclusion. Both, the infrastructure provisioning and component configuration are part of this. Infrastructure provisioning necessitates considerably less time than component configuration. Consequently, the scope of this thesis is limited to the provisioning and configuration of software, with infrastructure provisioning being excluded.

3.5.2 Concurrency

Concurrent processing can frequently facilitate the resolution of issues in a more efficient manner. For example, in the event that the CPU is the bottleneck, it may be beneficial to increase the number of CPU cores that are in use. Nevertheless, the problem must be capable of running concurrently in order to gain benefits out of concurrency. Numerous obligations necessitate sequential execution. Consequently, it is impossible to parallelize such scenarios, as user creation must always occur prior to its use in the configuration of other components. Second, the virtualization platform's constraints may necessitate an increase in CPU resources to optimize parallelization. However, this is not always feasible.

Regrettably, concurrency is not solely determined by the CPU. The main memory must be accessed by a greater number of CPU cores when they are operating in parallel, resulting in increased memory bandwidth and capacity requirements.

This data is essential for the purpose of conducting an analysis and evaluation. In order to ensure that the results are reproducible, it is necessary to measure these metrics and meticulously document the process.

3.5.3 Evaluation Platform

A platform that can quantify these parameters is necessary for the implementation of solutions and the execution of benchmarks. Ansible is the applicable platform for the

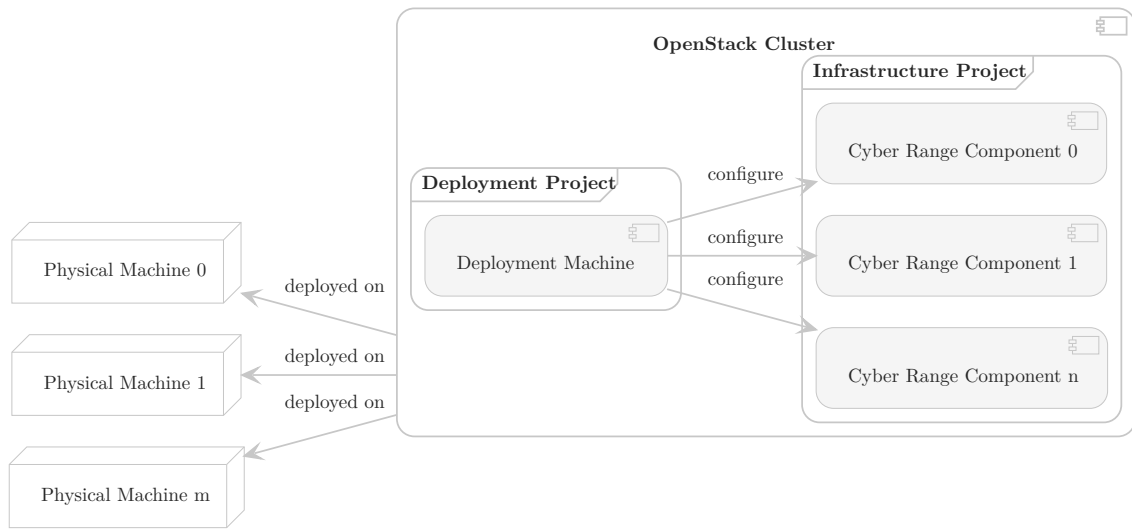


Figure 3.3: Infrastructure Deployment Architecture

implementation and verification of these patterns. This is motivated by the fact that Ansible is the most broadly used tool for performing deployments on platforms that necessitate full virtualization, such as a cyber range, as per Guerriero et al. [34]. The AIT Cyber Range is built upon this technology stack, so it is used to implement, test, and verify the findings of this work.

3.5.4 Setup

In an effort to replicate and comprehend the outcomes, the benchmarks must be executed within a predetermined environment. All benchmarking operations are conducted on the AIT OpenStack cluster. This guarantees that nothing relies on a development machine that would be exclusive to one individual.

Documenting the specifications of the OpenStack cluster provided by the AIT is crucial, as every machine operates within it. Each of the five physical nodes in the OpenStack cluster is equipped with a 512 GB main memory and two Intel Xeon Silver 4214R 2.4 GHz, 12C/24T CPUs, as illustrated in fig. 3.3.

Within this cluster, there are typically two distinct types of machines. The initial group consists of the machines that are included in the cyber range infrastructure project, as illustrated in fig. 3.3. Their requirements are found in the Terraform modules in the Infrastructure as Code repository and differ depending on the exercise.

The deployment machine is the type of machine that is employed to execute the Ansible playbooks. The deployment machine, which is the flavor with the highest performance in this cluster, is assigned 16 cores and 16 GB of main memory by the OpenStack cluster. However, in certain instances, alternative types of machines may be employed, such as

when specific software necessitates unique hardware specifications. The specification will be referenced in such instances.

The deployment time can be quantified in the following manner, regardless of whether improvements are implemented:

1. Adjust Deployment Size
2. Provision Infrastructure with Terraform
3. Provision Software using Ansible
4. Destroy Infrastructure with Terraform
5. Repeat Provisioning

In order to calibrate the infrastructure deployment for accurate scalability measurement, it is imperative to adjust the deployment size. The AIT Cyber Range necessitates the modification of Terraform modules to reflect the appropriate number of deployed machines.

The provisioning of the infrastructure is essential to ensure that all non-configured machines are accessible for software provisioning. This makes this step necessary after each deployment size adjustment.

Ansible's software provisioning is the primary component of this advancement. This part is the most time-consuming and is intended to be optimized in this work. This is the only process within the work that is evaluated in terms of time expenditure, i.e., the provision of infrastructure is not taken into account.

The time measurement is completed upon the accomplishment of the software provisioning, rendering the deployed infrastructure obsolete. To prevent the measurement from being influenced by previous software provisioning repetitions, the entire infrastructure is destroyed following the software provisioning process using Terraform.

Subsequently, this procedure is executed five times with identical input sizes in order to identify and rectify measurement discrepancies. This process will be repeated if the input size is adjusted.

Once the implementation is complete, the results must be assessed in order to demonstrate potential enhancements.

3.5.5 Measuring Process

The software provisioning stage, which corresponds to the Ansible infrastructure, is the primary focus of the AIT Cyber Range measurement. The Ansible infrastructure is composed of numerous playbooks and stages that are anticipated to exhibit significant differences in performance and require differing amounts of time.

It is crucial to measure the deployment process in a precise manner during the measurement process to understand which components of the deployment require the most time. Understanding them enables one to determine which tasks require more time and which require less. This serves to delineate the initiatives that require the most attention.

3.5.6 Result Presentation

The data results of the measuring process must be presented in a manner that is more easily comprehensible. In the context of deployment times, the selected presentation is a line chart that illustrates the necessary deployment time in relation to the number of deployed clients. The charts can be found in section 4.1. It is crucial to present the raw data in this manner, without any interpretation or analysis, to allow the reader to exclusively collect the objectives and conduct the interpretation independently.

However, the data is not particularly significant on its own. It is necessary to interpret the results in order to obtain any conclusions.

3.5.7 Analysis and Interpretation

An analysis that can be conducted with minimal effort is the deployment time per client. This outcome illustrates the deployment's capacity to accommodate an increasing number of clients. The deployment is actually scaling as intended when this value remains constant regardless of the number of clients or is even reciprocal with more clients.

By attempting to incorporate the benchmark's data into functions such as linear, polynomial, or exponential functions, this concept can be further developed. For this method, these functions are configured to accommodate data only up to the run with the highest number of clients. The final result will be attempted to be extrapolated by this function and compared to the actual result. In this manner, the optimal function can be identified, and a strict boundary can be established.

To acquire further results, it is advantageous to evaluate the benchmarks against an object. Benchmarking is the primary method employed in this work to establish a reference benchmark for comparison purposes prior to the implementation or enhancement of the actual pattern. If this is accomplished, the benchmarks can be replicated in the exact same manner, but with the new implementation.

There are several methods by which the results of these two benchmarks can be compared. Initially, it is possible to compare the absolute difference between these two benchmarks. By doing so, it may be feasible to determine whether a pattern has resulted in improvements in scalability. Secondly, it is possible to ascertain whether the class of functions in which the benchmarks are most appropriately positioned is equivalent. This knowledge can be employed to determine in advance whether a particular solution will exhibit significantly different scalability characteristics when deployed with a significantly greater number of clients than in the benchmarks. Last, there may be a specific point in the line-chart where the two lines intersect, which is referred to as a *break-even point*.

The border is marked by the fact that a solution may be faster to deploy for a limited number of clients, despite the fact that it scales less well than another solution. It is crucial to be aware of the break-even point, as it assists in determining which solution to employ in which circumstance.

In certain instances, it may not be feasible to directly compare the results to the reference chart, as certain patterns are dependent upon others. If this is the case, the dependency has to be outlined. Furthermore, this should be isolated, as the intermediate pattern may have already impacted the measurement in some capacity. The isolation may be feasible by means of the precise separation of deployment tasks.

3.6 Communication

The communication of the problem statement, along with the corresponding solution and evaluation, will take place in the course of this thesis. In particular, it is the answering of the research questions defined in section 1.3.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Current State

This chapter tracks the current state of the AIT Cyber Range as described in section 3.1. This is required to identify which focus should be put on during the literature research and the pattern selection.

4.1 Current Deployment Time in the AIT Cyber Range

To gain a more comprehensive understanding of the relevance of patterns and the metrics that must be estimated, the evaluation platform is initially benchmarked. This aids in estimating the metrics' score and helps to identify irrelevant patterns early. Each diagram represents the deployment time of a playbook either cold or warm in dependence of the amount of clients as explained in section 3.1.

In order to mitigate the likelihood of being influenced by outliers, each measurement is repeated five times as suggested by Henning et al. [38]. Beside the outliers, every diagram displays the most time consuming Ansible tasks. To show the development of the deployment time, the diagrams include linear and quadratic regressions of the total time.

The AIT Cyber Range is utilized to test this with 0-40 clients in five-step increments.

The most significant results out of the cold diagrams are listed below. Warm benchmarks are not that relevant for this work besides of a few cases which are elaborated later.

- Bootstrapping (fig. 4.1) scales linearly but takes the most time out of all playbooks.
- Clients (fig. 4.3) takes not as much time as bootstrapping, but scales quadratically.
- User management (fig. 4.5) scales linearly, however with a significant factor.

4. CURRENT STATE

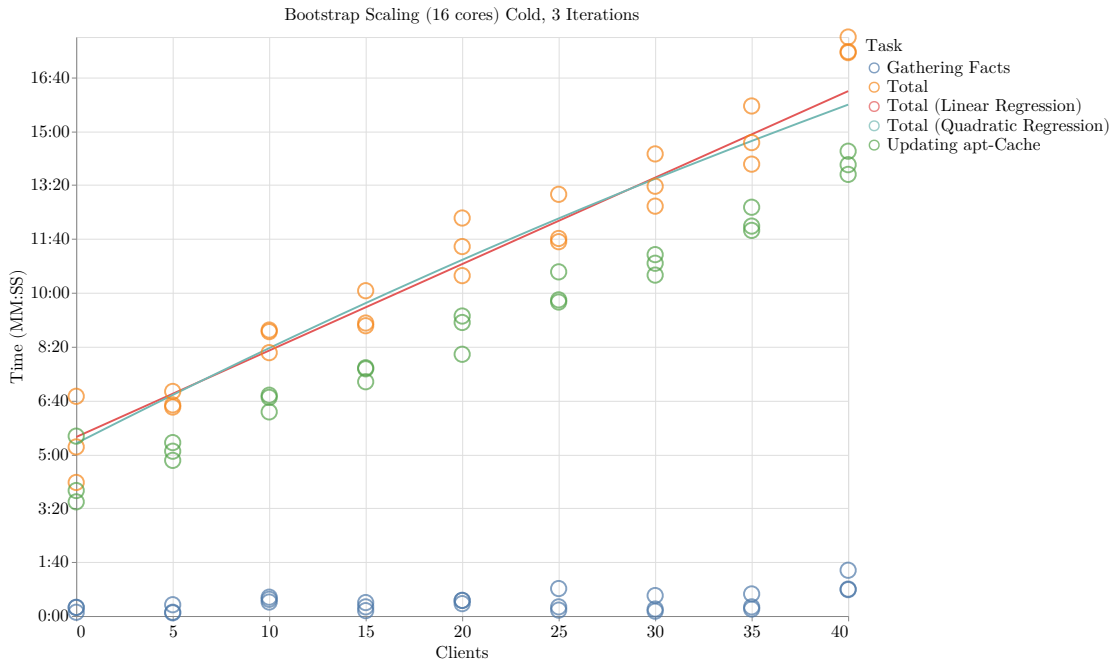


Figure 4.1: Bootstrap Scaling Cold

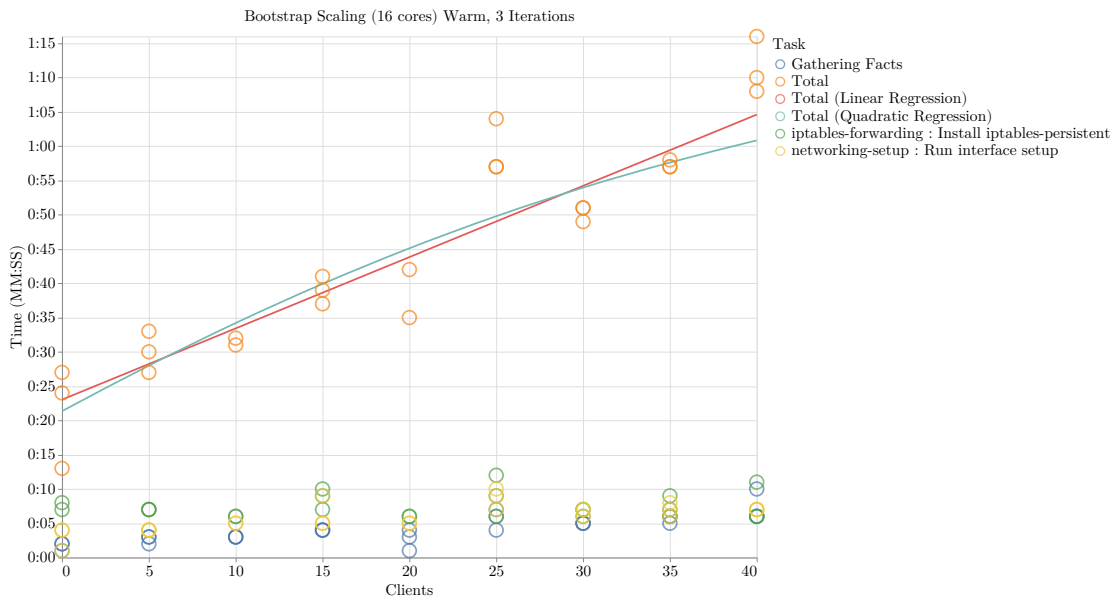


Figure 4.2: Bootstrap Scaling Warm

4.1. Current Deployment Time in the AIT Cyber Range

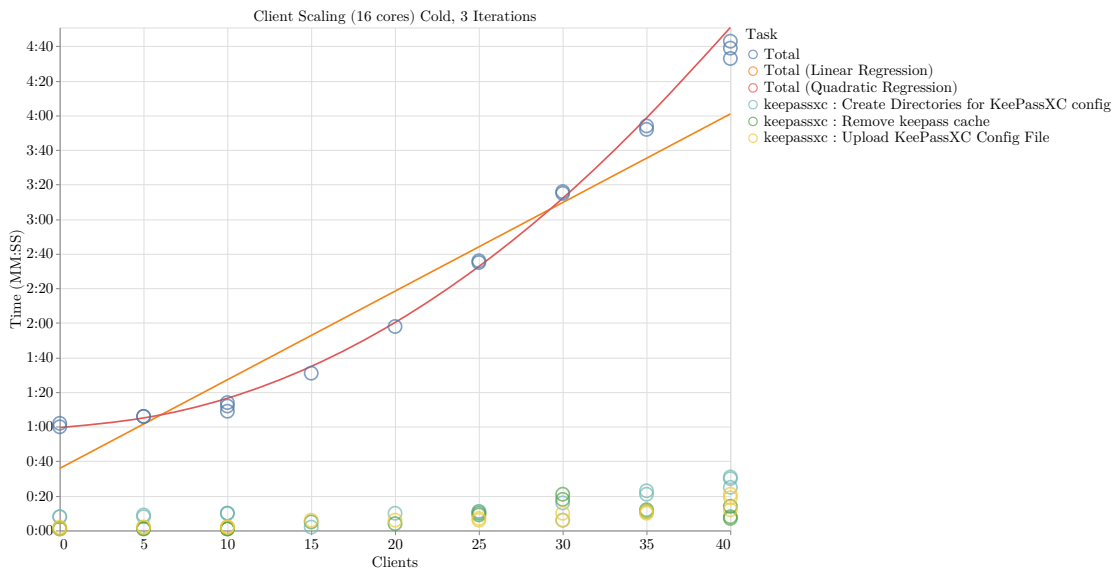


Figure 4.3: Client Scaling Cold

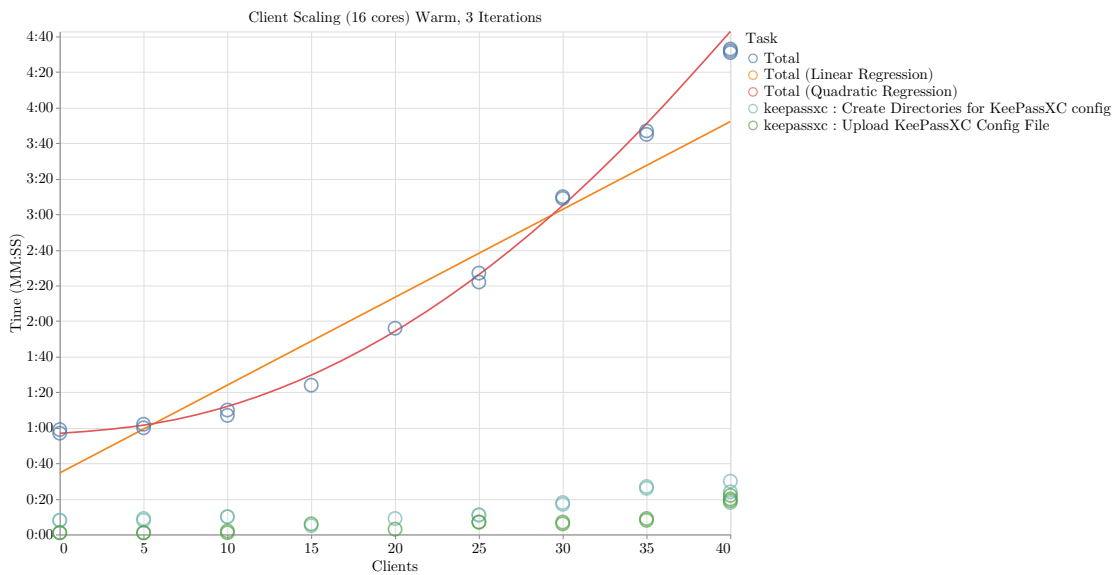


Figure 4.4: Client Scaling Warm

4. CURRENT STATE

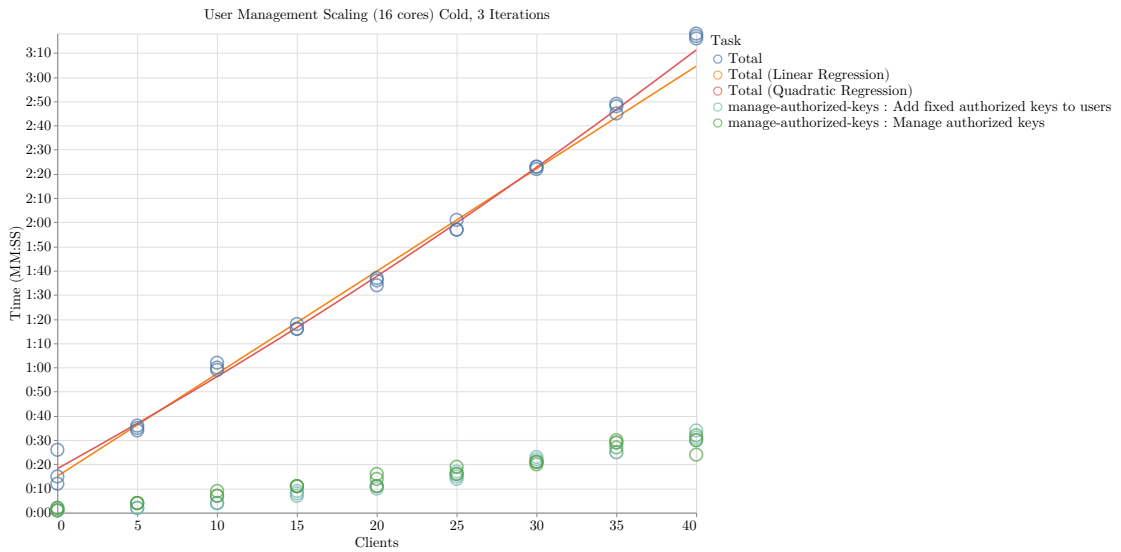


Figure 4.5: User Management Scaling Cold

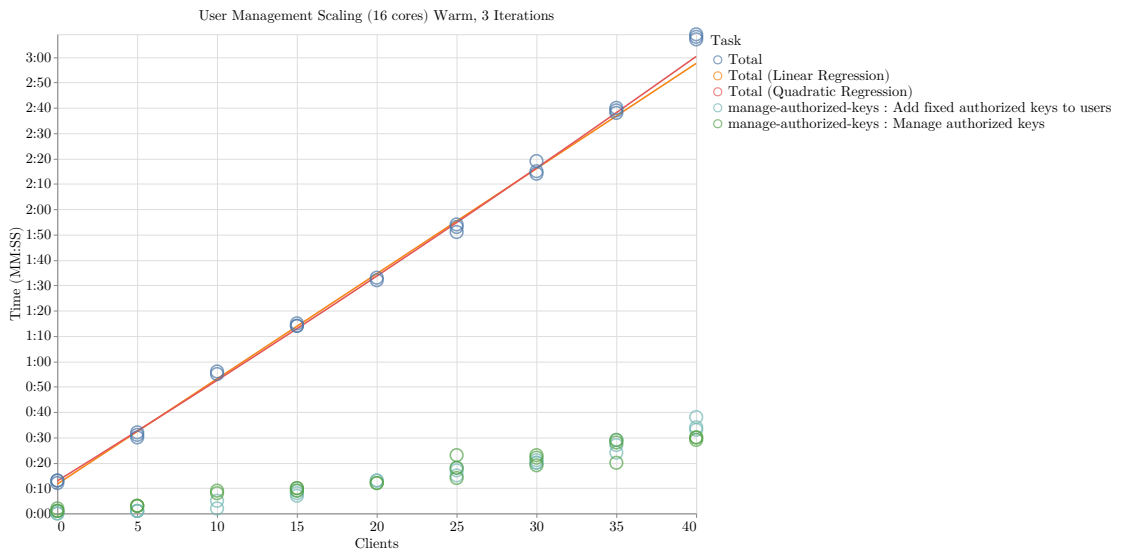


Figure 4.6: User Management Scaling Warm

4.1. Current Deployment Time in the AIT Cyber Range

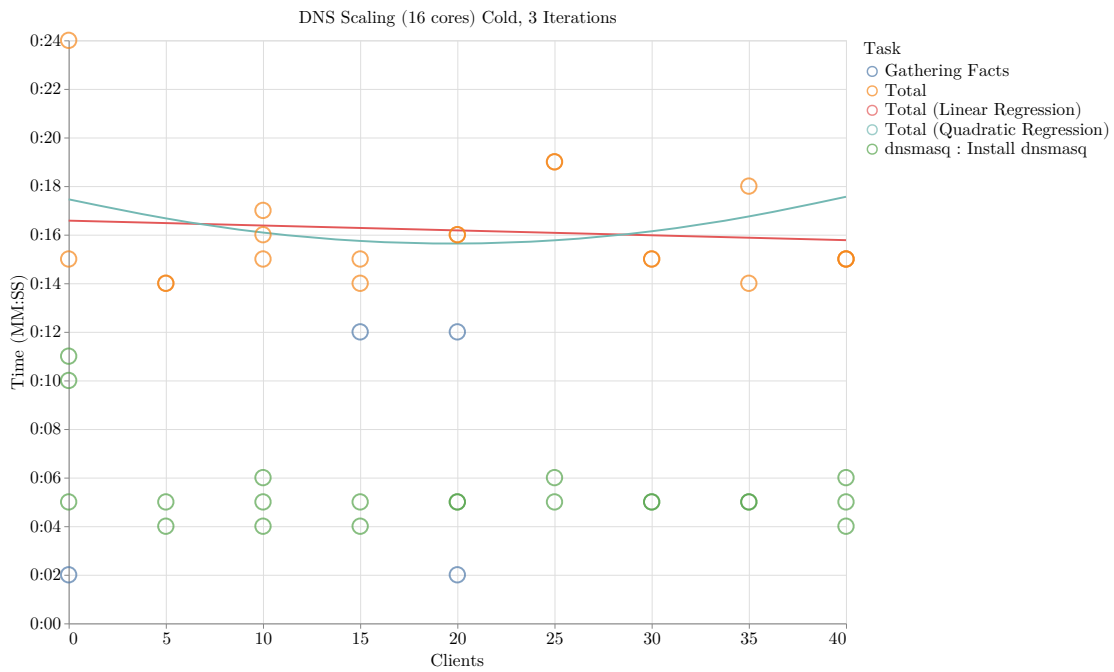


Figure 4.7: DNS Scaling Cold

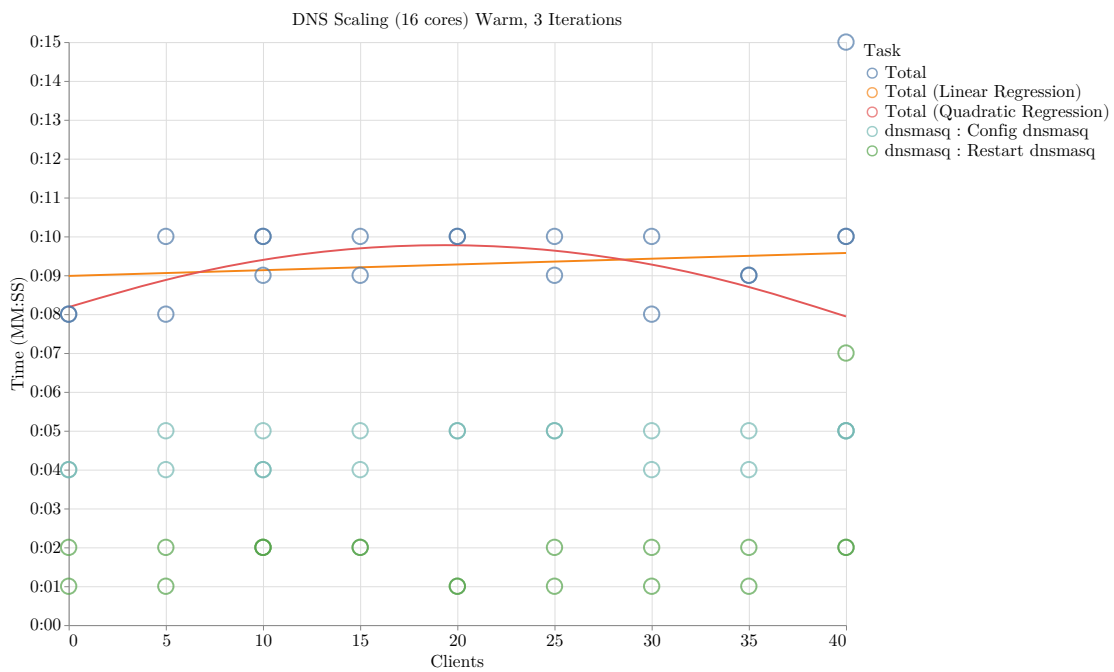


Figure 4.8: DNS Scaling Warm

4. CURRENT STATE

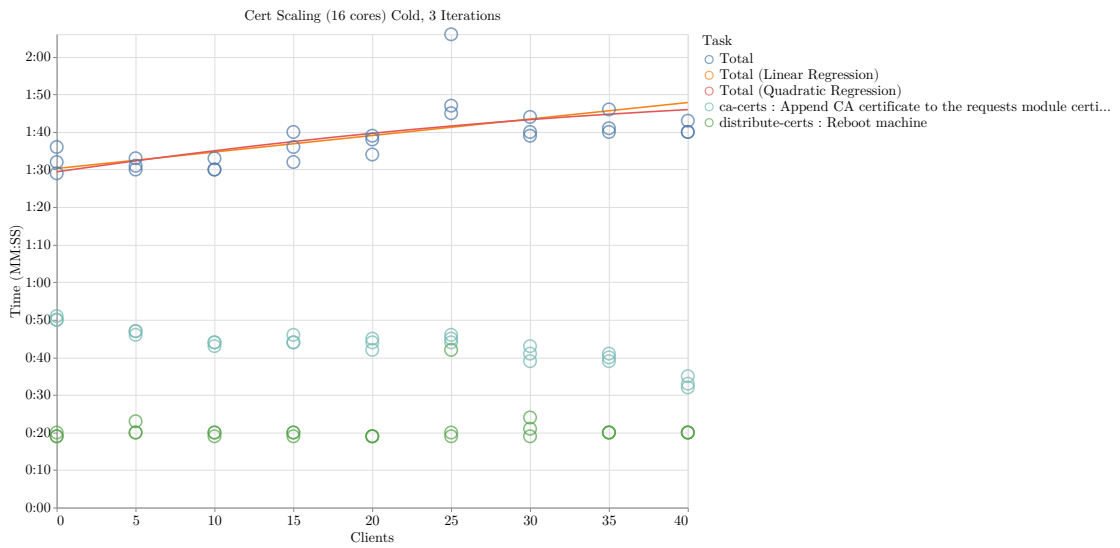


Figure 4.9: Certificate Scaling Cold

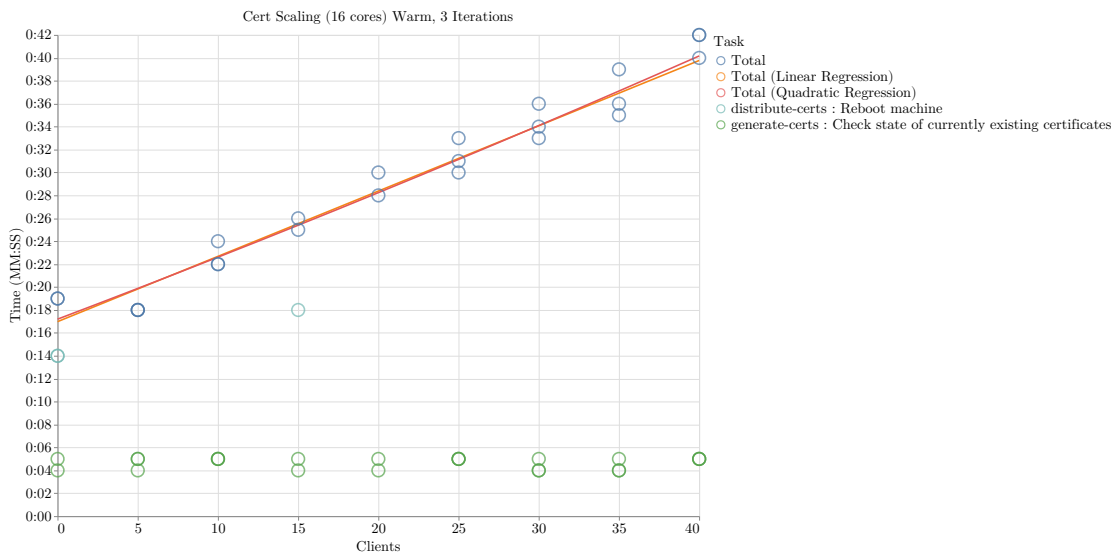


Figure 4.10: Certificate Scaling Warm

4.1. Current Deployment Time in the AIT Cyber Range

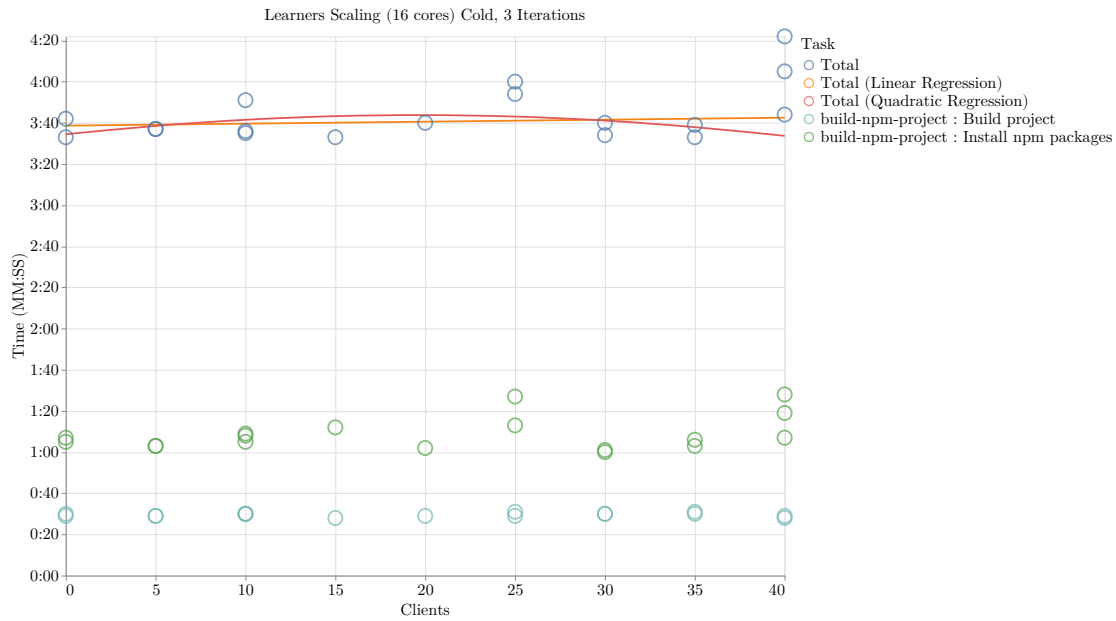


Figure 4.11: Learners Scaling Cold

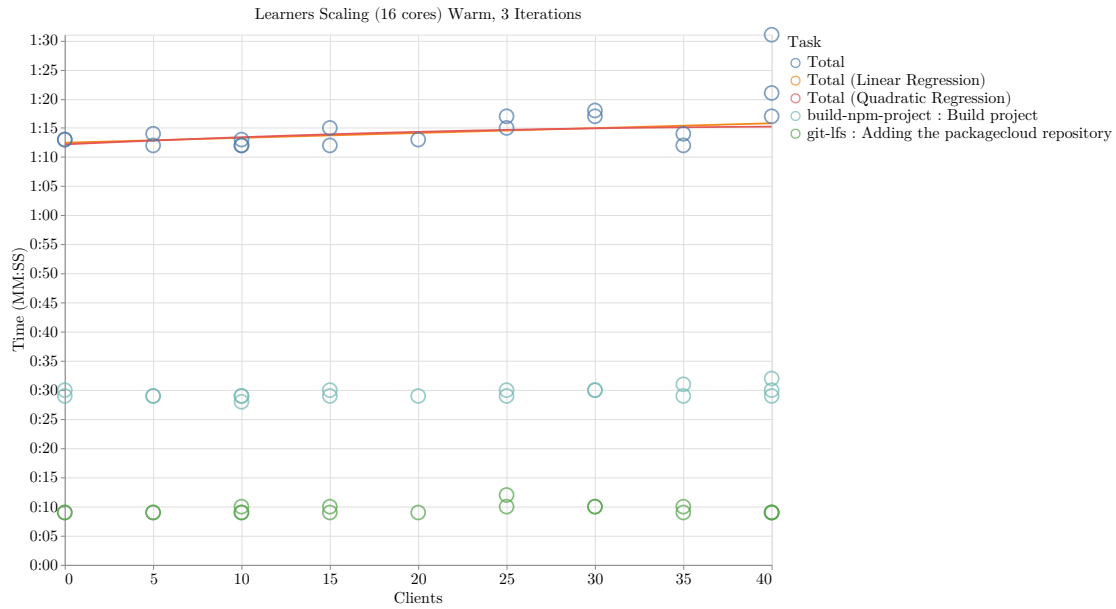


Figure 4.12: Learners Scaling Warm

- Due to being constant and only remaining under 30 seconds, DNS (fig. 4.7) is negligible.
- Certificate (fig. 4.9) scales linearly, however with a lower factor than User management.
- Learners (fig. 4.11) takes a significant amount of time, however, it remains constant.

The most significant outcomes of the benchmarks are the prolonged duration of the bootstrap playbook and the scalability issues of the client playbook. Therefore, these will receive the most attention in the subsequent sections.

4.1.1 Comparison of Cold and Warm Benchmarks

In general, it can be observed, that despite some diagrams such as fig. 4.3 and fig. 4.5 are showing increasing deployment times in dependence of the clients, their warm counterparts do as well. This indicates two possibilities:

1. The deployment workloads are not as heavy as expected. Instead, all the organizational work Ansible performs such as checking the current state, managing inventories or scheduling tasks is much more sophisticated and takes much more time than the actual work on the machines.
2. Ansible's idempotence checks do not work as expected which results into configuring the same things which are already done. The cause for this could be insufficient checks in the Ansible roles by being not careful enough, or by technical limitations which prevent such idempotence checks.

This indicates that measurements which effect the scheduling of the deployment rather than the deployment-workload itself can be assumed to have more influence on the deployment time.

4.2 Takeaways

The current state of the AIT Cyber Range allows to conclude the following takeaways:

- Concentrate on organizational patterns rather those effecting the deployment workload itself. This is important while executing the pattern selection.
- The most significant scaling issues are evident in fig. 4.1 and fig. 4.3, which require a significantly longer time to execute as more clients are deployed. This takeaway has the most effect on both, the pattern selection and during planning the concrete implementations.

- Conversely, there are components of the deployment, including fig. 4.7 and fig. 4.11, that are immune to the quantity of deployed clients. This has to be considered when skipping patterns which are within the k-top in the rank but are expected to be inconvenient for the evaluation.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Candidate Pattern Selection

The pattern selection process is important for identifying patterns in the literature that minimize the scalability issues that have been identified. Conversely, prior to initiating the filtering procedure, there are several prerequisites that must be fulfilled. The initial step was to conduct benchmarks of the current AIT Cyber Range deployment. Based on the results of chapter 4, this enables the subsequent filtering process to determine which patterns to prioritize, thereby assigning them a higher score in the pattern ranking. Second, metrics must be defined in order to establish a ranking based on their potential usefulness. The patterns can then be searched for in the literature, and the filtering process can begin.

5.1 Architectural Requirements of a Cyber Range

The architectural requirements of cyber ranges must be clarified in order to determine the appropriate selection process and metrics. The following are the primary architectural requirements for cyber ranges in general [46], the AIT Cyber Range and thus relevant for this work:

1. Adaptability is required to give cyber ranges the flexibility to change the exercise scenario. The more adaptable a cyber range is, the less maintenance effort is necessary for future scenarios.
2. The scenario has to be built in a way such that is configurable for participants. They have to solve tasks where components are misconfigured.
3. Of a security perspective, cyber ranges have to ensure that only users have access to resources they actually should have to in order to prevent participants are impacting each other or even cheating.

4. Cyber ranges are customized and deployed per exercise. This makes it necessary to reduce the deployment time as much as possible.

5.2 Finding Patterns from the Literature

Before starting to find patterns from the literature, a query has to be formulated to ensure reproducible results. The query endeavors to identify literature that addresses the scalability challenges associated with infrastructure deployments and proposes solutions. As the primary objective is to decrease the deployment time as it is implied by RQ₁ in section 1.3, a portion of the inquiry pertains to benchmarking on that particular subject.

The primary objective of the initial round is to identify literature that has already addressed the issue of scalability in infrastructure deployment through the use of patterns:

(infrastructure deployment \vee infrastructure provisioning
 \vee infrastructure as code \vee iac \vee configuration management)
 \wedge scalability \wedge patterns \wedge performance \wedge benchmark

The results of the search of the first three result pages in Google Scholar, which were identified as applicable for resolving scalability issues in infrastructure deployment, are presented in the following list.

Literature	Annotation
Claudio A. Ardagna et.al. [7]	Describes patterns of the view of a PaaS provider. Cyber ranges are no PaaS provider.
Daniel Sokolowski [76] Sören Henning et.al [38]	No patterns, but how to benchmark scalability of cloud-native applications. States that ≤ 5 repetitions are required.
Jukka Koskelin [50] Alberto Avritzer et.al. [10]	Interesting for performance when scaling compute resources in virtual environments, and Multi-Client Component.
Rui Han et.al. [35]	Scale applications at run-time more efficiently. Also interesting for Multi-Client Component.
Rajesh Komar [49]	Comparison of *aaS. Reference for Multi-Cloud patterns. requirements
Muhammad Ehsan Rana et.al. [9]	No patterns, but scalability in cloud applications. Reasoning why architectural patterns are useful for cyber range scalability.
Christoph Fehling et.al. [23]	Cloud and patterns but no scalability.

The following are the limitations of the search query and the existing literature: One significant challenge encountered during the literature research was the identification of literature that accurately reflected the intended definition of scalability in this work. Although the majority of the literature defines scalability as the capacity to accommodate an increased volume of the same workload during runtime, the term “scalability” in this work is intended to refer to the ability of the same resource to scale appropriately without focusing on runtime issues.

Another constraint is that the majority of the literature discovered pertains to the portability of cloud deployments. Their primary concern is whether it is straightforward to transition from existing systems to a cloud system or to transition from one cloud provider to another. Nevertheless, neither of these are significant issues in this thesis.

However, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications* is a source that contains a significant number of patterns [25]. This book has already compiled numerous patterns that are suitable for cloud computing. It describes the problem that each pattern solves, the solution that must be implemented, and the considerations that must be taken. Consequently, a significant amount of literature review has already been conducted to identify new patterns, shifting the focus to this source.

5.3 Pattern Refinement

Prior to the execution of any pattern selection, it is necessary to locate patterns. Table 5.1 enumerates every pattern that was discovered in the literature. The scope of this work does not encompass the explanation and elaboration of all patterns that are invoked. Nevertheless, they are thoroughly delineated in the citation.

Given the size of this list, a pre-selection was conducted in order to minimize the amount of work required for the actual ranking. The pre-selection process involves estimating the pattern’s relevance to cyber ranges. There is no actual quantification of the relevance, as this is an estimation. However, it should serve as a preliminary starting point for reducing the number of patterns.

5.3.1 Relevance

The estimation of the relevance of patterns is influenced by a few factors that do not adhere to a strict quantification process. The reasons a pattern would not apply to a cyber range are described by the following Kick-out Factors (KOFs):

- (KOF 1) The pattern affects resources, such as long-time storage, that are not necessary for cyber ranges in general.
- (KOF 2) The pattern is intended for distributed applications. Typically, cyber ranges are not required to manage state synchronization or maintain consistency across multiple components.

- (KOF 3) The pattern is not applicable for deployment time but development or runtime instead.
- (KOF 4) The pattern introduces safety or resilience features such as watchdogs. The challenge to solve such issues is part of the scenarios and must be solved by the participants.
- (KOF 5) The pattern is designed to address the issue of managing large data streams. Cyber ranges do not handle massive data streams, aside from very specific scenarios.

The results of which patterns were not filtered by the KOFs and will therefore be examined further are shown by table 5.1. A short description of each is listed below:

Application Component Proxy Provide a proxy which enables access to resources in an environment with restricted access. This allows to isolate the restricted environment from external access.

Provider Adapter Implement a generic interface for cloud providers to decouple software from specific vendor implementations. Prevents vendor lock-in.

Content Distribution Network Cache frequently accessed resources within the private network. This reduces the internet usage, reducing cost and increasing the speed of streaming resources which are frequently required.

Loose Coupling Reduce the dependencies between components as much as possible. This increases scalability and fault tolerance while reducing network load.

Processing Component Distribute heavy workloads across multiple processing components. This enables the ability to scale out elastically on heavy workloads.

Managed Configuration Provide a centralized configuration store for all components. This allows to control all components in a coordinated way without the need of re-configuring them while increasing consistency.

Elasticity Management Process Implement a component which automatically scales application instances according to the current workload in both directions. This helps to manage intense workloads when required while also remaining cost efficient.

Dedicated Component Instantiate components multiple times despite being multi-tenancy capable. This increases the isolation of the components to protect sensitive data.

User Interface Component Decouple the user interface components from the rest of the environment in a way such that it can be configured and customized completely independent from the rest of the application.

Name	Relevant/KOF(5.3.1)	Rejection Reason	Source
Elasticity Management Process	✓		[26]
Content Distribution Network	✓		[45]
Loose Coupling	✓		[86]
Processing Component	✓		[40]
Managed Configuration	✓		[83]
Dedicated Component	✓		[89]
Multi-Component Image	✓		[2]
Shared Component	✓		[89]
Tenant-Isolated Component	✓		[89]
Provider Adapter	✓		[25]
User Interface Component	✓		[86]
Transaction-Based Processor	✓		[14]
Application Component Proxy	✓		[23]
Hybrid Processing	1	Cyber ranges change the deployment platform frequently and components are hard to reuse.	[25]
Restricted Data Access Component	2, 4	Not relevant for deployment time, nor tackling scalability.	[27]
Distributed Application	2	Cyber ranges do not have necessarily applications to split.	[4]
Stateful Component	2	State management has no focus in cyber ranges.	[29]
Stateless Component	2	State management has no focus in cyber ranges.	[27]
Batch Processing Component	5	Big data processing is not the focus of cyber ranges.	[77]
Data Access Component	5	Big data processing is not the focus of cyber ranges.	[28]
Data Abtractor	5	Big data processing is not the focus of cyber ranges.	[24]
Idempotent Processor	2	Pattern is about consistency, not scalability.	[41]
Timeout-Based Message Processor	2	Message-oriented middleware are no core functionality of cyber ranges.	[82]
Message Mover	2	Message-oriented middleware are no core functionality of cyber ranges.	[17]
Compliant Data Replication	2	The scalability issues are not caused by accessing data.	[41]
Integration Provider	2	The cyber range deployment is done in a single environment.	[17]
Elasticity Manager	3	Workload is predictable and fully depends on scenario/participant registrations.	[72]
Elastic Load Balancer	3	Workload is predictable and fully depends on scenario/participant registrations.	[3]
Elastic Queue	3	Workload is predictable and fully depends on scenario/participant registrations.	[36]
Watchdog	4	Participants are responsible for the infrastructure.	[22]
Feature Flag Management Process	3	Does not effect deployment scalability.	[1]
Update Transition Process	4	Downtimes are allowed during the deployment.	[44]
Standby Pooling Process	3	Only effects provisioning speed during runtime, but not deployment time.	[39]
Resiliency Management Process	4	Should be done by the participants.	[56]
Two-Tier Cloud Application	5	Cyber ranges do not have large data streams.	[85]
Three-Tier Cloud Application	5	Cyber ranges do not have large data streams.	[85]
Hybrid User Interface	3	Not relevant for deployment time, nor tackling scalability.	[37]
Hybrid Data	5	There is no 'data' scalability problem in the cyber range.	[71]
Hybrid Backup	1	Cyber ranges are short living and do not have backups.	[73]
Hybrid Backend	3	This is a runtime problem.	[42]
Hybrid Application Functions	3	This is a runtime problem.	[25]
Hybrid Multimedia Web Application	5	There are not much large multi-media files.	[87]
Hybrid Development Environment	3	Only relevant during development, not deployment.	[51]

Table 5.1: Pattern Relevance

Multi-Component Image Instead of forging dedicated VM images for every different component, create a single image with all required software parts installed fitting all components. This allows each component to switch its task to another one which is required more urgently during runtime without the need of any reconfiguration.

Shared Component Share components across multiple tenants. This reduces the deployment effort while saving cloud resources.

Tenant-Isolated Component Share components across multiple tenants similar to Shared-Component. The difference is that the isolation prevents tenants from influencing each other.

Transaction-Based Processor Implement a mechanism to ensure consistency while processing messages such as deployment instructions.

5.4 Pattern Ranking

As stated in the chapter 3, this work establishes a ranking for two reasons. Firstly, to enhance the efficiency of the pattern selection process, and secondly, to generate a ranking as a result of this investigation.

5.4.1 Pattern Ranking Metrics

The pattern ranking itself uses a scoring system based on estimation. A few categories are defined by this system, and they can be either fulfilled by a pattern or not. Upon the subject's fulfillment, the pattern is awarded points on a scale of 1 to 5. The final score of the pattern is determined by the sum of the points; the higher the sum, the better.

Risk This metric considers the security requirements of a cyber range. Potential risk vectors are cloud API access to cyber range components, single-point of failure or mechanisms that reduce the isolation of different companies or teams. When a pattern is expected to have no risk, it gains more points in this category.

Impact Whether a pattern affects many components. In this event, the likelihood of a more severe outcome increases. For instance, a pattern will receive a higher score if it encompasses clients, web servers, and application servers.

Dependencies When numerous components are dependent on a pattern, it may require a significant amount of effort to integrate it into existing platforms. As a result, the objective is accomplished in this instance when the dependencies are minimal. Things such as the implementation of a new protocol or the extensive reorganization of the architecture lead to a lower score.

Maintainability The less effort required to maintain a pattern after it has been implemented, the more straightforward it is to employ. This point might involve the

Name	Risk	Impact	Dependencies	Maintainability	Flexibility	Sum
Application Component Proxy	3.0	4.0	4.3	4.3	4.3	20.0
Content Distribution Network	4.2	3.6	3.6	4.2	4.4	20.0
Managed Configuration	4.5	3.8	3.0	3.4	3.8	18.5
Loose Coupling	3.8	3.0	3.4	3.6	3.6	17.4
Provider Adapter	3.5	3.8	3.0	2.8	3.2	16.2
Elasticity Management Process	3.2	3.7	2.8	3.2	2.4	15.3
Transaction-Based Processor	3.8	3.2	3.2	2.5	2.0	14.8
User Interface Component	4.0	3.2	3.0	2.8	1.5	14.5
Processing Component	4.0	2.6	2.8	1.6	2.6	13.6
Dedicated Component	3.2	2.0	3.2	2.6	1.6	12.6
Tenant-Isolated Component	2.2	2.2	2.4	1.8	1.6	10.2
Multi-Component Image	2.5	1.8	1.6	1.2	2.6	9.7
Shared Component	1.5	1.8	2.2	1.8	1.6	8.9

Table 5.2: Pattern Metric Expert

pattern’s actual structure. Declarative deployment methodologies that necessitate numerous variables and templates are perceived as less maintainable. This metric is adversely affected by the introduction of numerous new components or the risk of diverging configuration.

Flexibility The probability of a pattern being applicable in a broader range of concrete platforms increases as its abstraction increases. This metric is lowered by requirements for highly specialized protocols or highly specific configuration.

5.4.2 Pattern Ranking Results

This yields two outcomes when applied to the pertinent patterns: table 5.2, which displays the estimations for each metric per pattern. Furthermore, each metric is visually represented by an axis in fig. 5.1 and fig. 5.1. The pattern in that instance has scored more points the more the axis is covered resulting in a larger area the more point a pattern achieved.

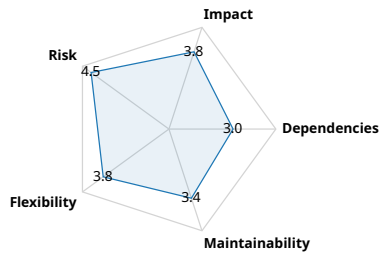


(a) Application Component Proxy

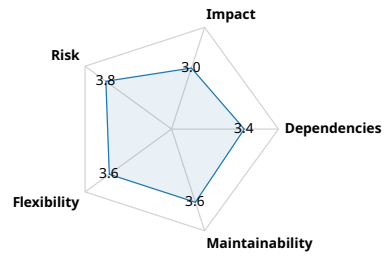
(b) Content Distribution Network

Figure 5.1: Radar Diagrams of Metric Estimations per Pattern

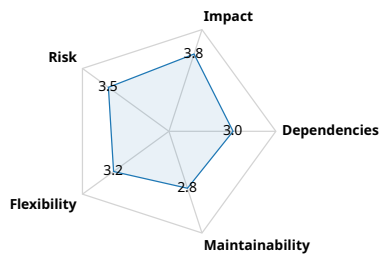
5. CANDIDATE PATTERN SELECTION



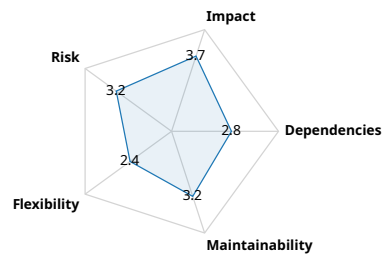
(c) Managed Configuration



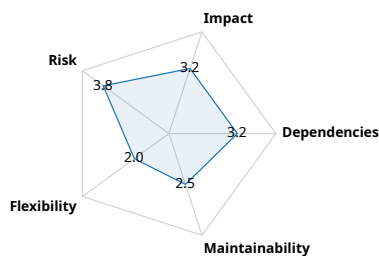
(d) Loose Coupling



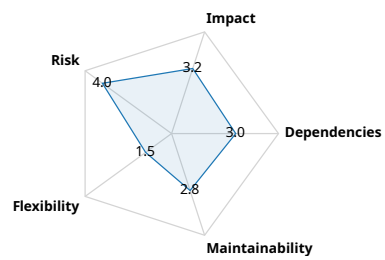
(e) Provider Adapter



(f) Elasticity Management Process

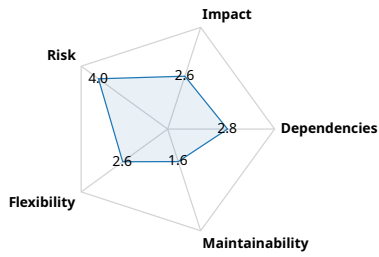


(g) Transaction-Based Processor

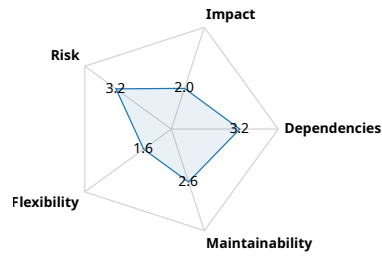


(h) User Interface Component

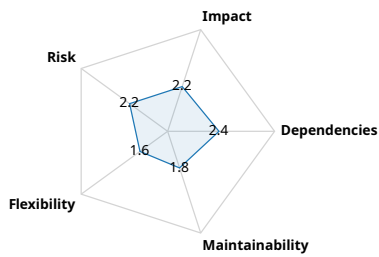
Figure 5.1: Radar Diagrams of Metric Estimations per Pattern



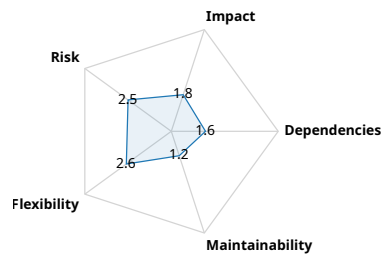
(i) Processing Component



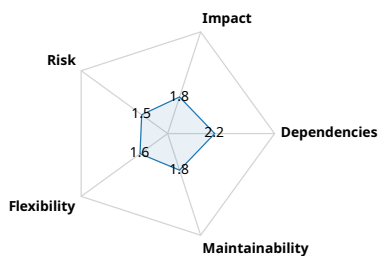
(j) Dedicated Component



(k) Tenant-Isolated Component



(l) Multi-Component Image



(m) Shared Component

Figure 5.1: Radar Diagrams of Metric Estimations per Pattern

5.5 Final Pattern Selection

As indicated by the findings in table 5.2, the *Application Component Proxy*, *Content Distribution Network* and *Managed Configuration* are the three patterns with the highest scores. This indicates that these three should be implemented in order to achieve the highest efficiency as determined by the metrics. Nevertheless, we have omitted a few patterns for the following reasons:

Managed Configuration Red Hat Ansible Automation Platform [67] and Ansible AWX [5] are the two options that can be used to implement this, as the evaluation platform utilizes Ansible. However, given how widely both are extending Ansible, this would assess a technological advancement rather than an architectural change, diminishing its significance in comparison to other patterns.

Loose Coupling Although this pattern may have a substantial effect on scalability, it is challenging to determine whether it has been implemented or not. This is more of a guide for long-term code base maintenance because, in practice, superfluous dependencies are eliminated when they are found. This pattern is omitted because of the uncertainty surrounding whether it can be deemed implemented or not.

Provider Adapter This pattern is omitted since substituting the cloud provider would potentially result in completely different benchmarks which would not be comparable with the current state.

As a consequence, the subsequent patterns are to be implemented:

1. Application Component Proxy
2. Content Distribution Network
3. Elasticity Management Process

Implementation of Candidate Patterns

This chapter contains the assessment of the patterns that were selected in chapter 5. The division is based on patterns, and for each pattern, the implementation process is described, along with reasons for choosing certain implementation details and problems that came up during implementation. Finally, each pattern is benchmarked and the most significant results are presented. To understand the individual benchmark diagrams, we advise to refer to section 3.1.

6.1 Implementation Decisions of Candidate Patterns

The patterns considered to have the most potential are presented in section 5.5. Since this part of the work has to make concrete decisions on the particular implementation of the pattern, they get new names assigned at this point. This helps to distinguish between the patterns obtained from the literature and the patterns where the implementation decisions were made.

- Application Component Proxy → Proxy Jump Component
- Content Distribution Network → Package Cache Component
- Elasticity Management Process → Multi-Client Component

6.2 Proxy Jump Component

The *Proxy Jump Component* defines a proxy jump host to be implemented between the deployment host and the nodes to configure. This makes it possible to isolate the nodes

to configure from any public networks and shift the network load from a potential weak link between deployment host and private network to a potential strong link between the proxy jump and the private network.

6.2.1 Implementation

The AIT Cyber Range uses Ansible for the software provisioning which relies on SSH for machine communication. This enables the opportunity to implement the *Proxy Jump Component* with SSH.

SSH has the capability to use proxy jumps natively. However, when implemented, the deployment process of the AIT Cyber Range became unstable. Attempts to establish connections were unsuccessful or timed out frequently.

To address this issue, the proxy jump host necessitated the following configuration options:

MaxSessions Specifies how many sessions can be multiplexed through one connection. This is important for proxy jumps and was increased from 10 to 5000, since a cyber range deployment consists of machines in the magnitude of hundreds.

MaxStartups Specifies how many connections SSH remains open which are still not authenticated. This is important when deploying lots of machines at the same time, since the deployment host opens connections to lots of machines simultaneously. Therefore, this was also increased to 5000.

However, the AIT Cyber Range already follows the *Proxy Jump Component* pattern. Nevertheless, it is one of the patterns with the most potential. Therefore, the evaluation of the impact of the pattern is still relevant for this work. To be able to identify the impact of the pattern, benchmarks are recorded without using any of the proxy jump hosts, meaning the pattern was unimplemented in this case. This means that an improvement in this case would actually confute that this pattern makes improvements.

6.2.2 Benchmark Results

As it can be observed by the diagrams of the benchmarks, the numbers of the provisioned clients is limited to 25 due to stability issues discussed in section 7.1.2 when the pattern is not implemented.

- Figure 6.1 shows that at 25 clients the total deployment time increases from $\approx 7:50$ to $\approx 12:20$ (fig. 4.1) on average, which is an increase in time of more than 50%. The amount of time required for 0 clients remains the same, meaning that only the scaling factor has increased.



Figure 6.1: Bootstrap Scaling Cold (without Proxy Jump Component)

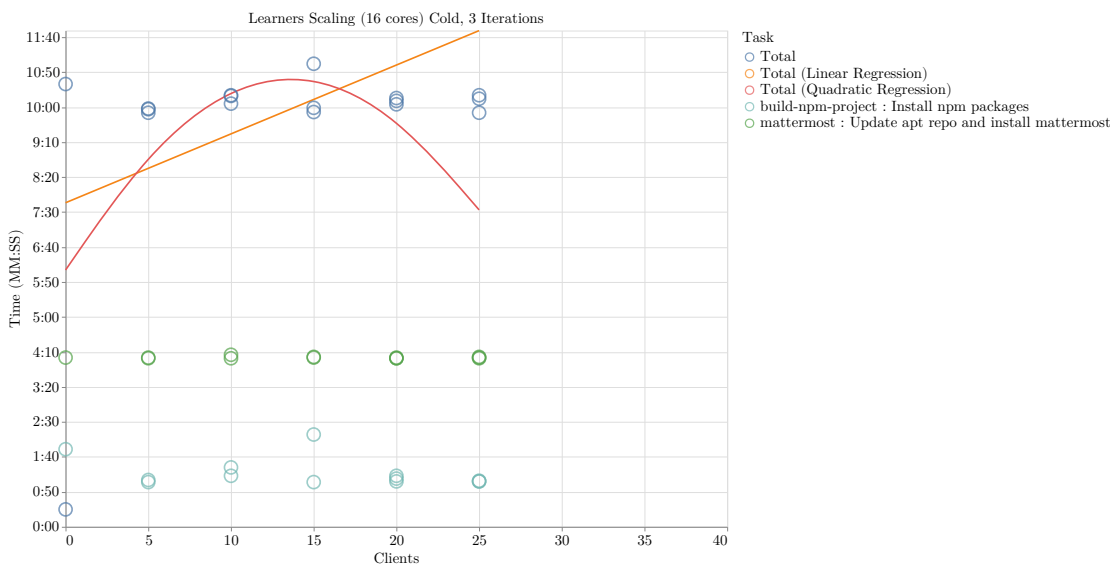


Figure 6.2: Learners Scaling Cold (without Proxy Jump Component)

- Figure 6.2 displays the increase of the total deployment time at 25 clients from $\approx 3:40$ constantly (fig. 4.11) to ≈ 10 on average. This is an increase of $\approx 170\%$ independent of the amount of clients.

Results without significant changes:

- Figure A.1 *Client Scaling Cold (without Proxy Jump Component)*
- Figure A.2 *User Management Scaling Cold (without Proxy Jump Component)*
- Figure A.3 *DNS Scaling Cold (without Proxy Jump Component)*
- Figure A.4 *Certificate Scaling Cold (without Proxy Jump Component)*

6.3 Package Cache Component

The *Package Cache Component* aims to deploy a node to cache packages for other components. This increases the speed of the installation of frequently required packages.

6.3.1 Implementation

The *Package Cache Component* pattern aims to provide a central machine which acts as a cache for Linux packages. Since the AIT Cyber Range uses Ubuntu for mostly all machines, the package cache to evaluate is an apt-cache.

There are multiple options available to implement an apt-cache. The first one chosen was Apt-Cacher-NG [31] since it is a caching proxy specifically designed for apt. This should offer the advantage in better optimization such as removing old package versions earlier to reduce disk space or preloading known dependencies in contrast to pure HTTP proxies. However, when implemented it resulted in unstable behavior namely corrupting packages, being unresponsive to clients or even crashes. This behavior is due to bugs in Apt-Cacher-NG, such as segmentation faults, concurrency issues or hash miscalculations [16].

Another proxy for apt caching is Squid-deb-proxy [32] which is a pre-configured Squid proxy [78] with emphasis on apt caching. This solution turned out working and was integrated as Ansible playbooks in the AIT Cyber Range for evaluation. The important part is that it is configured at the very beginning to ensure software updates and installations can already utilize the proxy during deployment time.

6.3.2 Benchmark Results

Results with significant changes:

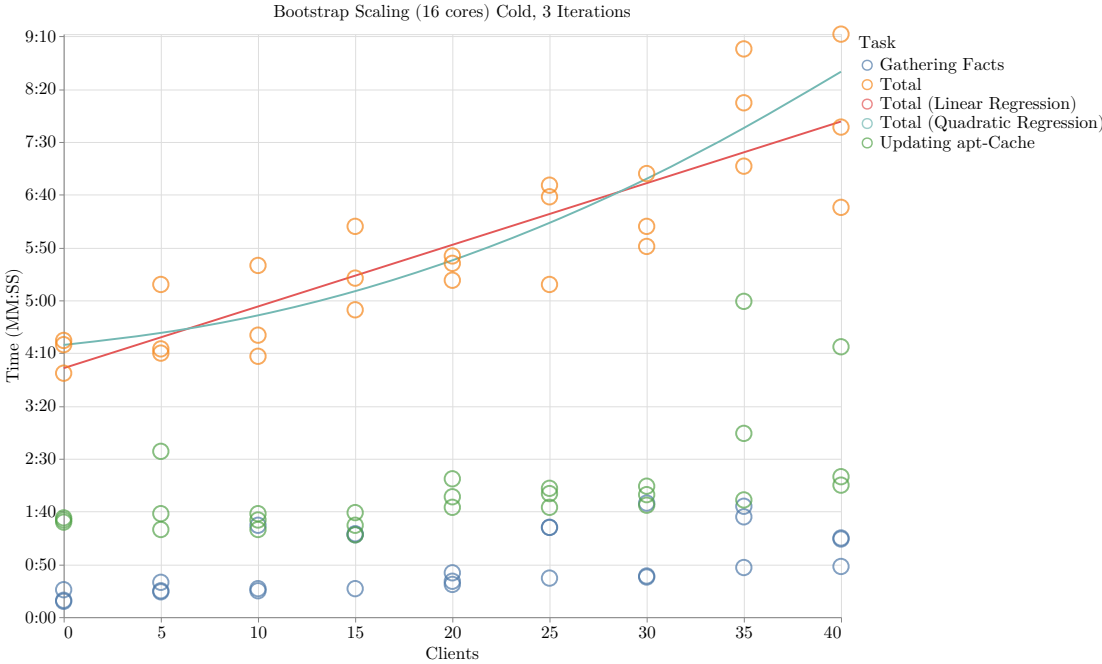


Figure 6.3: Bootstrap Scaling Cold (Package Cache Component)

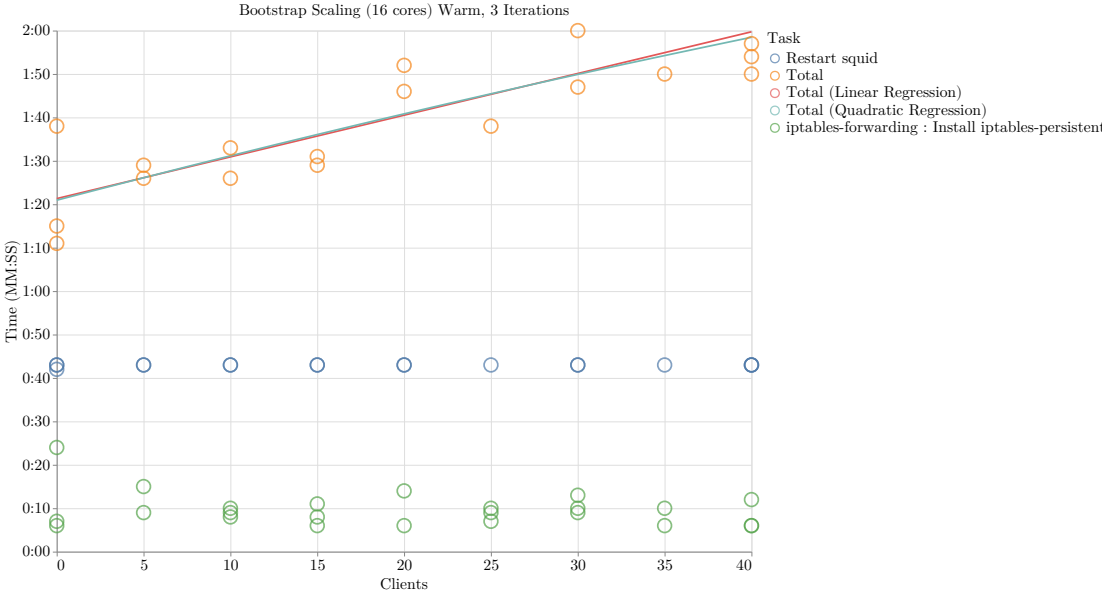


Figure 6.4: Bootstrap Scaling Warm (Package Cache Component)

- Figure 6.3 shows that the total deployment time of 0 clients decreases from $\approx 5:10$ to $\approx 4:10$ on average, which is a reduction of $\approx 20\%$ while at 40 clients the time decreases from $\approx 17:00$ to $\approx 7:50$ on average, which is a reduction of $\approx 50\%$. However, the fluctuation of the minima and maxima has increased.
- Figure 6.4 shows that the total deployment time of 0 clients increases from $\approx 0:20$ to $\approx 1:20$ on average, which is an increase in time of $\approx 300\%$ while at 40 clients the time went up from $\approx 1:10$ to $\approx 1:50$ which is an increase of $\approx 60\%$.

Results without significant changes:

- Figure A.5 *Client Scaling Cold (Package Cache Component)*
- Figure A.6 *Client Scaling Warm (Package Cache Component)*
- Figure A.7 *User Management Scaling Cold (Package Cache Component)*
- Figure A.8 *User Management Scaling Warm (Package Cache Component)*
- Figure A.9 *DNS Scaling Cold (Package Cache Component)*
- Figure A.10 *DNS Scaling Warm (Package Cache Component)*
- Figure A.11 *Certificate Scaling Cold (Package Cache Component)*
- Figure A.12 *Certificate Scaling Warm (Package Cache Component)*
- Figure A.13 *Learners Scaling Cold (Package Cache Component)*
- Figure A.14 *Learners Scaling Warm (Package Cache Component)*

6.4 Multi-Client Component

The *Multi-Client Component* encourages to instantiate multiple client sessions on a single component. This reduces the amount of necessary client nodes to deploy.

6.4.1 Implementation

The initial step was to establish an environment that would enable a single host to manage multiple client sessions. This approach involved the utilization of Incus [58], a containerization platform that is a fork of LXD. The client image was relocated to this new component in order to facilitate client instantiation. The noVNC platform is now required to establish a connection with the *Multi-Client Component*, rather than all client machines.

The noVNC platform for participation expects that each client is directly accessible over VNC. However, by hosting multiple client sessions on a single host, this is not the case

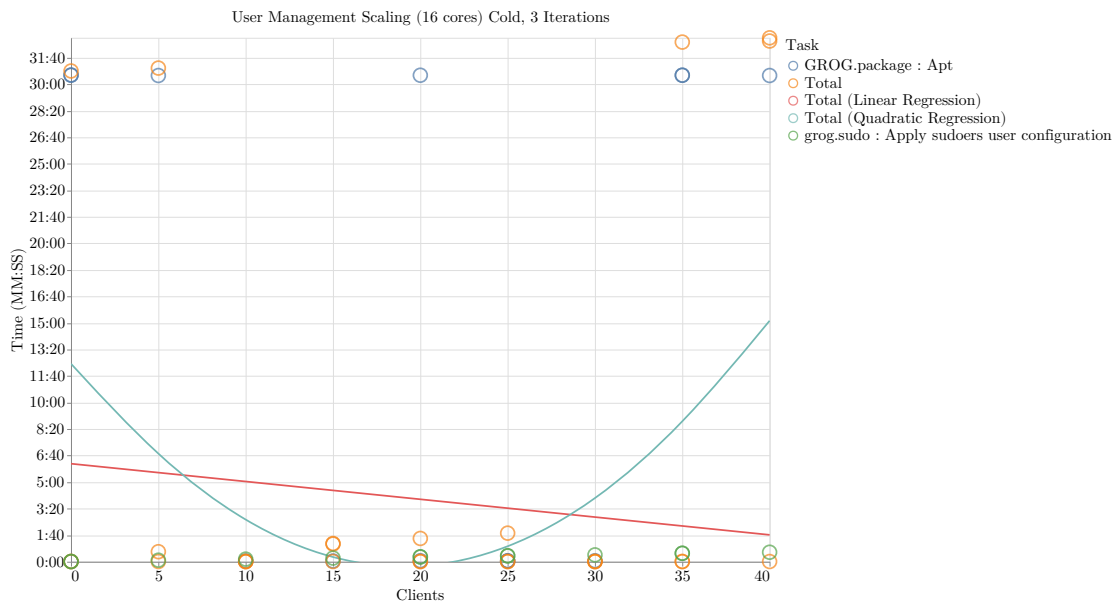


Figure 6.5: User Management Scaling Cold (Multi-Client Component)

anymore. To solve this issue, the *Multi-Client Component* uses the socket activation feature from systemd. The socket activation allows the *Multi-Client Component* to listen on a single TCP port. A shell script is initiated when the noVNC platform establishes a connection to the port, which leads to the establishment of a new client instance and the transfer of the connection, thereby rendering the multi-session transparent to the noVNC platform.

The client is instantiated on demand, which results in a slight delay. However, this process is completed in a matter of seconds.

A major challenge of this implementation was to configure options in the new client image provided for the participants on the containerization platform within the *Multi-Client Component* after the infrastructure provisioning. Due to the fact that the *Multi-Client Component* generates client instances, the dynamic Ansible inventory and OpenStack are unable to monitor them.

The solution was to utilize the `ansible.builtin.add_host` module to include all containers returned by Incus. The disadvantage of this method is that the deployment performance may be negatively impacted by the necessity of transitioning from *free* to *linear* in the Ansible strategy.

6.4.2 Benchmark Results

All benchmark results suffer from rather high fluctuations. The reason for this is that the implementation of the Multi-Client Component turned out to be quite unstable. The deployment suffered from hanging and lots of network time outs making a viable

benchmark impossible to perform. Figure 6.5 is the highest outlier, however, all others were unstable as well:

- Figure A.15 *Bootstrap Scaling Cold (Multi-Client Component)*
- Figure A.16 *Client Scaling Cold (Multi-Client Component)*
- Figure A.17 *DNS Scaling Cold (Multi-Client Component)*
- Figure A.18 *Certificate Scaling Cold (Multi-Client Component)*
- Figure A.19 *Learners Scaling Cold (Multi-Client Component)*

6.5 Summary of Findings

To summarize, the *Package Cache Component* is able to achieve the most reduction in deployment time. The *Proxy Jump Component* increases the deployment slightly but makes it possible to deploy more than 25 clients. The *Multi-Client Component* causes major stability issues preventing any significant statement about the deployment time can be claimed.

Discussion

The first part of this chapter discusses the patterns that were implemented in chapter 6. The discussion commences with the scalability improvements for each pattern, which are further separated into the expected effects, the scalability improvements that the pattern actually implemented, and a rationale for whether or not the anticipated effects were realized. Following this, a discussion of the side effects and additional details to consider for each pattern is conducted. The discussion concludes by providing thoughts on the methodology that emerge during the demonstration and evaluation and addressing the research questions from section 1.3.

7.1 Proxy Jump Component

The *Proxy Jump Component* introduces a jump host which acts as a tunnel between two nodes. This allows to hide sensitive nodes from public networks. The implementation decisions are described in section 6.2.

7.1.1 Scalability Improvements

The *Proxy Jump Component* is expected to have an impact on every stage of the software provisioning since every stage of the deployment uses SSH connections. Especially stages with lots of little tasks are expected to improve, since the session-multiplexing should reduce the overhead which is created by connection handshakes.

As it can be observed in section 6.2.2, when implemented, the *Proxy Jump Component* causes the deployment time to remain the same or to increase. However, only the cold bootstrap and the cold learners are effected by this degradation. In both cases, the complexity class remained the same.

One reason why it scales worse when implemented is that the network latencies between multiple nodes sum up. Given that the network latency between hosts A, B and C is

equal, when using host B as a proxy jump between host A and C, the latency essentially doubles. Additionally, more compute resources are required to handle the proxied network packets.

7.1.2 Eradication of Client Limits

The number of clients that can be deployed on the AIT Cyber Range is restricted to 25 when the *Proxy Jump Component* is not implemented, as illustrated in section 6.2.2. The reason for this is that network issues occurred when the number of floating IPs in use on OpenStack is increased. Particularly, even though they were provisioned successfully, the OpenStack API only returns a maximum of roughly 25–30 floating IPs (a variable quantity). Thus, the *Proxy Jump Component* is a solution to this issue, as demonstrated.

7.1.3 Man-in-the-Middle

As stated in the manpage `SSH(1)` [79]:

Agent forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the agent's UNIX-domain socket) can access the local agent through the forwarded connection. An attacker cannot obtain key material from the agent, however they can perform operations on the keys that enable them to authenticate using the identities loaded into the agent. A safer alternative may be to use a jump host (see `-J`).

It is crucial to either completely prevent agent forwarding or limit access to the jump host to individuals who are also granted equal privileges to access the target machines.

7.1.4 Floating IPs

The *Proxy Jump Component* pattern is advantageous for another reason, even though it increases the deployment time on the evaluation platform. Given that not all components must be accessible from the configuring host during the deployment, the necessity for IP addresses that are either public or, in the case of OpenStack, floating IPs, is diminished. Since both typically cost money, this implies a cost reduction when it is not necessary.

7.2 Package Cache Component

The *Package Cache Component* introduces a node which hosts cached packages for other nodes to decrease the required time for software installation and updates. The implementation decisions are described in section 6.3.

7.2.1 Scalability Improvements

Improvements are expected in stages such as bootstrapping, where lot of updating and software installation is being performed. Stages which do not manage software installations should not be affected by this such certificated, DNS or user management. This pattern is expected to show great benefits in environments where the internet bandwidth is limited, but the local area network is quite strong.

As it can be observed from section 6.3.2, the cold bootstrapping scales better when the *Package Cache Component* is implemented which is as expected, despite fluctuating a bit more in minima and maxima in relative but not in absolute. However, the warm bootstrapping scales a bit worse than before.

A possible explanation why the warm bootstrapping scales worse than before, is that the *Package Cache Component* introduces an extra step during the deployment. Normally, when the bootstrapping runs the second time, it is not expected that any software will be installed or updated. In contrast to the packages itself, the package databases have to be synchronized with the upstream nevertheless and the extra hop over the proxy is enough to worsen the scalability in this particular case. During the cold bootstrap, the benefit of caching the packages outweighs the proxy overhead.

7.2.2 Component Scope

The *Package Cache Component* as implemented in this work is part of the exercise infrastructure of the cyber range. This implies that the *Package Cache Component* is also deployed each time a scenario is deployed. Although this method guarantees the reproducibility of the benchmarks, it also reduces the potential for reducing the deployment time. In particular, if a long-running instance of the *Package Cache Component* were to exist independently of the exercise infrastructure, each package would be cached during the initial deployment. This decreases the deployment time for all subsequent deployments when employing comparable packages.

7.3 Multi-Client Component

The *Multi-Client Component* provides a platform which is able to host multiple clients at once, reducing the amount of clients to be deployed and maintained. The implementation decision are described in section 6.4.

7.3.1 Scalability Improvements

There is an expectation that the *Multi-Client Component* will reduce the deployment time of the stages that are primarily responsible for configuring clients. Nevertheless, it is anticipated that this pattern will result in an increase in the processing time required to configure the *Multi-Client Component*, as it is required to configure the containerization platform. The expected deployment time decreases as the number of clients that can

be instantiated on a single *Multi-Client Component* increases. This is because fewer machines to configure are necessary, and the client stages only need to be run once per containerization platform, despite hosting multiple clients.

While reducing the deployment time in a few cases, the scalability of the *Multi-Client Component* is completely unpredictable. This behavior is the case in every playbook as observed in section 6.4.2.

The *Multi-Client Component* pattern turned out to be unstable within the AIT Cyber Range deployment. The problem was that many times, Ansible was unable to reach the containers within the containerization platform. The reason for this is not clear, and it does not happen every time. This makes the pattern in this state not very fitting since it prevents automatic deployments.

7.3.2 Dynamic Client Scaling

The *Multi-Client Component* allows to provision the clients on demand when a participant requests access to one. A single client container instance has to be provisioned as normal for this to function. The client container simply clones itself whenever a participant requests a new client. This results in the infrastructure and software provisioning being required only for a single client, allowing the platform to scale to many clients with the restriction of the container platform's hardware resources. This on-demand principle solves the issue that only a fixed amount of participants can use the clients and enables a cyber range to work for an unpredicted amount of participants. The maximum amount of participants is limited by the resources the containerization platform can provide and the type of workload the participants cause.

7.3.3 Containerization Overhead

Although the clients share their CPU and RAM resources dynamically on the containerization platform in the *Multi-Client Component*, the containerization platform introduces another layer of abstraction, increasing the amount of required RAM and CPU resources. This overhead is caused by the inability to use shared libraries across containers as well as the isolation of the containers.

7.3.4 Shared Disk Images

The implementation of the *Multi-Client Component* utilizes the Btrfs file system for storing the client images. The on demand procedure of the containerization platform to provision new client instances utilizes the CoW capabilities of Btrfs resulting in two benefits:

1. Since only the difference of client disks is stored, less storage is used.
2. The on-demand procedure is significantly faster than the full client disk cloning process.

7.3.5 Isolation

Since the *Multi-Client Component* runs multiple clients only isolated via containerization technology, measurements are required to ensure proper isolation. While there exist solutions to achieve this even with root access within the container, cyber ranges require graphical sessions, which require elevated permissions. To at least ensure that participants cannot affect participants from other companies, it is recommended to use separate container hosts for each company.

7.4 Pattern Combination

One potential approach to enhance scalability is to simultaneously implement multiple patterns on a cyber range. For that, choosing to implement the k-top patterns according to the ranking may seem to be the most logical way.

7.4.1 Mutual Exclusivity

Although the patterns used in this work made it possible to implement them simultaneously, it may not always be the case. One reason is that this is simply not possible by design, as different design principles might conflict with each other, such as the *Shared Component* and *Tenant-Isolated Component*. Technical reasons could be another factor. The combination of the *Multi-Client Component* pattern and the *Elasticity Management Process* pattern is an example of this. This is due to the limitations of Ansible's ability to configure nodes behind multiple nested connections.

7.4.2 Similarity

The metrics only consider one pattern at a time. So, there is no penalty if similar patterns are ranked or even already implemented. This results into similar patterns will gain a similar score. All in all, this means that it might not always be viable to just implement the k-top patterns, instead it has also to be checked if amongst the k-top there are any similarities.

7.5 Implementation Effort

Although it was attempted to incorporate it into the metrics through *dependencies*, it is not always possible to fully comprehend the effort pattern that leads to a cyber range. The primary reason is that there are other factors that impact the implementation effort. To mention a few:

- One example is the complexity of the pattern which were not considered in the metrics at all.

- A pattern that has already been implemented may not be in conflict with another pattern that is being implemented. Nevertheless, it may require a significantly greater amount of effort to execute both.
- Some patterns exhibit more stability issues during deployment than others during the development cycle. These have to be mitigated in some way, increasing both, development and evaluation effort, since often these stability issues arise after intense and repeated deployments.
- The patterns employed for deployments are significantly less restrictive and leave more flexibility in terms of implementation than those employed in programming languages. Although this permits a wide range of implementations, it also necessitates more work in the planning and design stages.
- In contrast to the patterns employed in this work for deployment, patterns used in programming languages must typically be implemented on source code under the control of the implementor. Additionally, the variables and parameters of software components cannot be altered during deployments, which increases the effort required on cyber ranges.

7.5.1 Implementation Dependent Results

While the implementation of the patterns was rationalized and discussed in chapter 6, they are not the sole method of implementing the corresponding pattern. There may be discrepancies between the patterns in terms of scalability, as they themselves allow for some flexibility in determining the specific implementation. Demonstrate or refute these discrepancies by implementing the same pattern using distinct methodologies and comparing the results.

7.5.2 Technology Impact

Pattern implementations were required to undergo evaluation on an evaluation platform in accordance with the selected Design Science Methodology. The challenge is that the patterns may exhibit different behavior when implemented in a different environment since different platforms may possess distinct properties. These platform differences may include:

- Architectural design
- Scenario complexity
- Software choices of components and the cloud platform

7.5.3 Benchmark Clustering

The benchmark diagrams in this work are arranged according to the playbook that corresponds to them. In other words, there is a diagram for clients, another for user management, and another for the deployment of the noVNC platform. On the other hand, these playbooks frequently share identical tasks, which implies that they have an impact on multiple playbooks. Examples of these tasks include copying and sharing data, installing software packages, and more. While specific playbooks, including the DNS and Certs playbook, are already organized by task rather than host, the evaluation could be further developed in this direction. This has the advantage of allowing the effect of patterns to be directly observed in the tasks, rather than on a per-machine basis.

7.5.4 Benchmark Reliability

Given that the evaluation platform employs Ansible, it was necessary to establish a method for benchmarking the deployment time. While Ansible is capable of printing profiling information to the console, the formatting is not machine-friendly, necessitating the creation of a script to extract all the times. Nevertheless, the process of parsing text that is intended for human consumption is highly error-prone. Consequently, it is desirable for Ansible to be able to print the profiling information in a machine-readable format, such as CSV.

Additionally, the deployment process frequently fails or remains indefinitely suspended as a result of network issues that can be resolved by rerunning the script. Consequently, it is imperative to periodically monitor the benchmark's status and re-run it as needed. It is time-consuming to benchmark, and it could potentially affect the data, as re-running the deployment may be faster due to the fact that some tasks may have already been executed in the previous failed attempt.

These concerns could be resolved through the implementation of an appropriate benchmarking suite for Ansible.

7.6 Research Questions

With the support and evidence of this work, the research questions previously defined in section 1.3 can now be answered as follows:

RQ₁: To what extent can Pattern-based Deployment Models improve the scalability of Infrastructure as Code driven configuration architectures?

The deployment time can be reduced by 50% at least in certain cases such as shown in section 6.3.2. However, this may also cause increasing deployment times in other places. The reduction from $O(n^2)$ to a complexity class strictly less may be possible. A candidate for that is the *Multi-Client Component* which however caused unstable behavior.

RQ₂: Which metrics can be used to perform a ranking on patterns used in deployment models to perform a pattern selection process? Throughout this work *risk*, *impact*, *dependencies*, *maintainability* and *flexibility* were used to perform the pattern selection process to chose three out of 13 patterns. One of the three patterns was able to increase the scalability by reducing the deployment time. Two of the three patterns have beneficial side effects such as allowing more hosts to be deployed or reducing the network traffic.

RQ₃: Which measurements can be taken to improve the scalability of Infrastructure as Code driven configuration architectures beside pattern-based Deployment Models? A possibility is to optimize the network traffic shaping regarding fairness instead of prioritizing high throughput on the deployment platform. The effects of having no traffic shaping can be observed from fig. 6.3 where much fluctuation has been recorded. Another possibility is to evaluate how suitable the software provisioning tool is for the job and comparing that with other software provisioning tools. This observation is the results from comparing fig. 4.3 and fig. 4.4 where the latter takes almost the same time despite the workload is drastically less.

Conclusion

We investigated the requirements of cyber ranges in this thesis to devise a method for implementing architectural patterns to enhance the scalability of the deployment process. The methods describe the process of conducting a literature review and selecting patterns using metrics that were refined based on the current state of the AIT Cyber Range. The patterns were ranked by four cyber range experts based on these metrics. Afterward, the AIT Cyber Range was employed as an evaluation platform to implement the top three patterns (*Proxy Jump Component*, *Package Cache Component*, and *Multi-Client Component*) in order to illustrate the method's significance and to showcase the potential scalability enhancement that patterns can offer in cyber ranges. The patterns were assessed using benchmarks, which resulted in up to 50% reduction in deployment time. Finally, the discourse discloses the causes of patterns' limited scalability, including unbalanced network traffic overhead and caching overhead. Furthermore, additional adverse consequences, including the elimination of client limits and the reduction of compute resource costs, are addressed.

8.1 Future Work

The following key topics provide a starting point for future work:

Find new Patterns The number of patterns selected in this work is restricted. Nevertheless, there are undoubtedly additional patterns in the literature, and new ones will be designed. The next step is to identify patterns and evaluate them in accordance with the relevance scheme outlined in chapter 5.

Implement Additional Patterns To evaluate the metrics refined in chapter 5, three pattern implementations were produced. This serves as a proof of concept and serves as a starting point; however, the metrics should be tested on a wider range

of patterns. These patterns may be patterns already mentioned in this work that were not implemented or even patterns that were not mentioned in this work at all.

Maintenance In this work, three patterns were implemented and tested for scalability; however, their long-term properties were not evaluated; they were only estimated in chapter 5. Maintenance tasks that necessitate consideration include:

- Implement new features
- Software updates
- Add or remove components

Benchmarks

This appendix contains all benchmarks which were produced throughout this work but were not significant for any discussion.

A.1 Proxy Jump Component

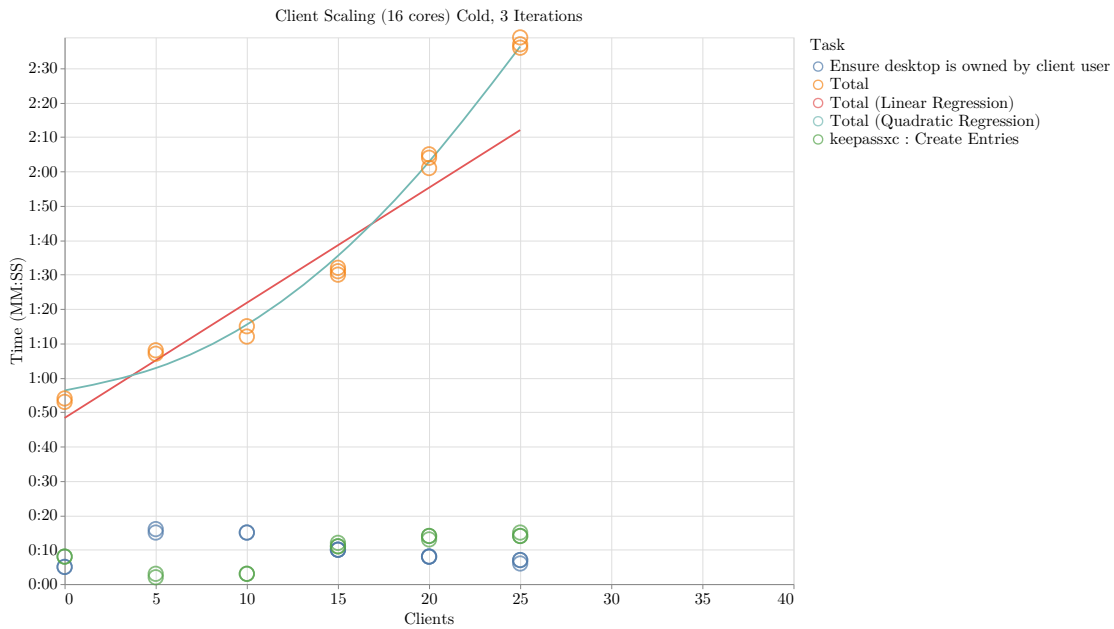


Figure A.1: Client Scaling Cold (without Proxy Jump Component)

A. BENCHMARKS

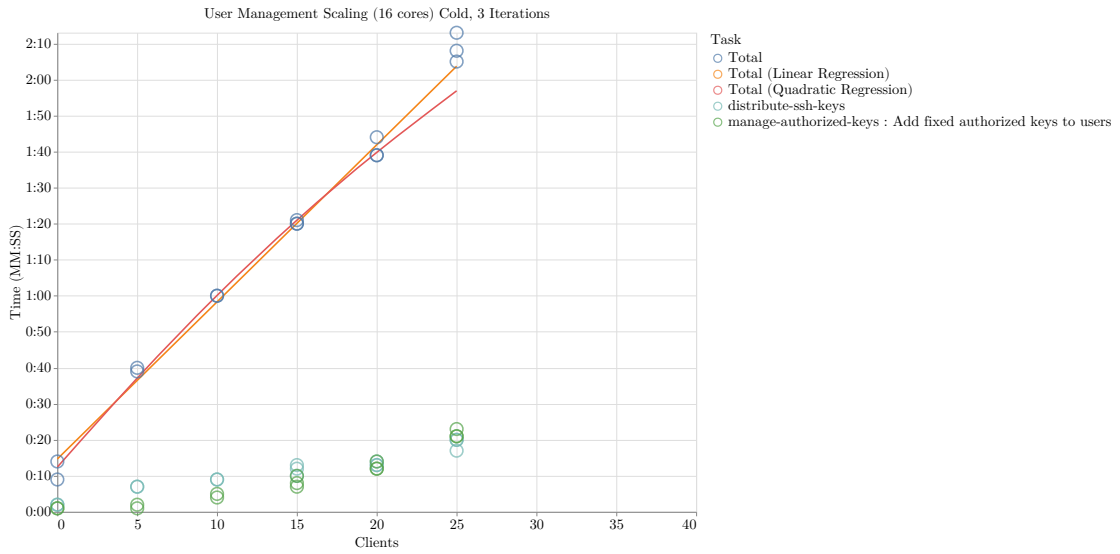


Figure A.2: User Management Scaling Cold (without Proxy Jump Component)

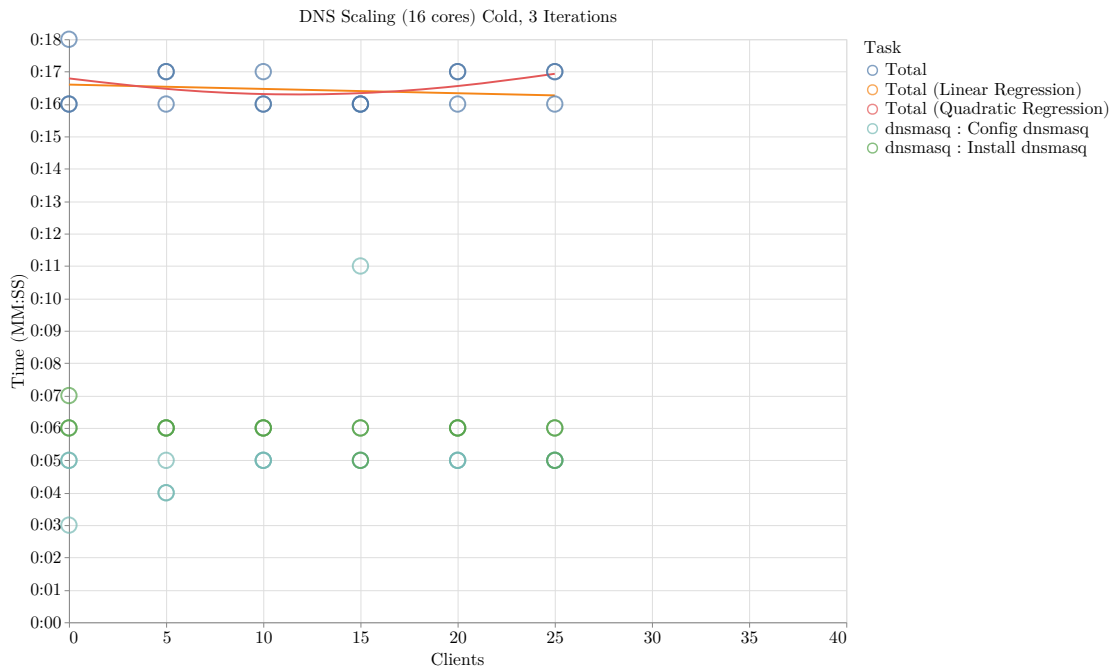


Figure A.3: DNS Scaling Cold (without Proxy Jump Component)

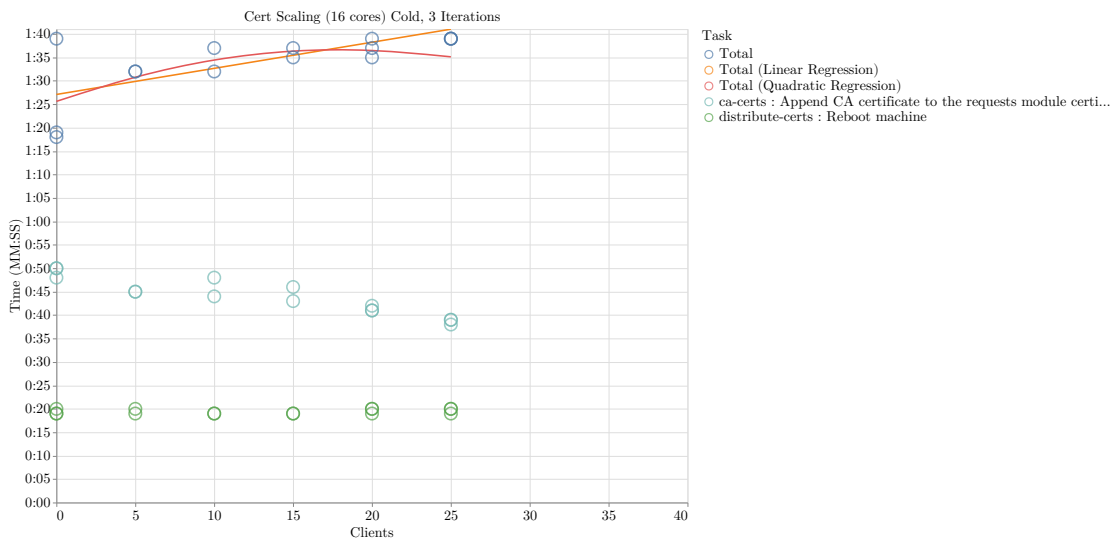


Figure A.4: Certificate Scaling Cold (without Proxy Jump Component)

A.2 Package Cache Component

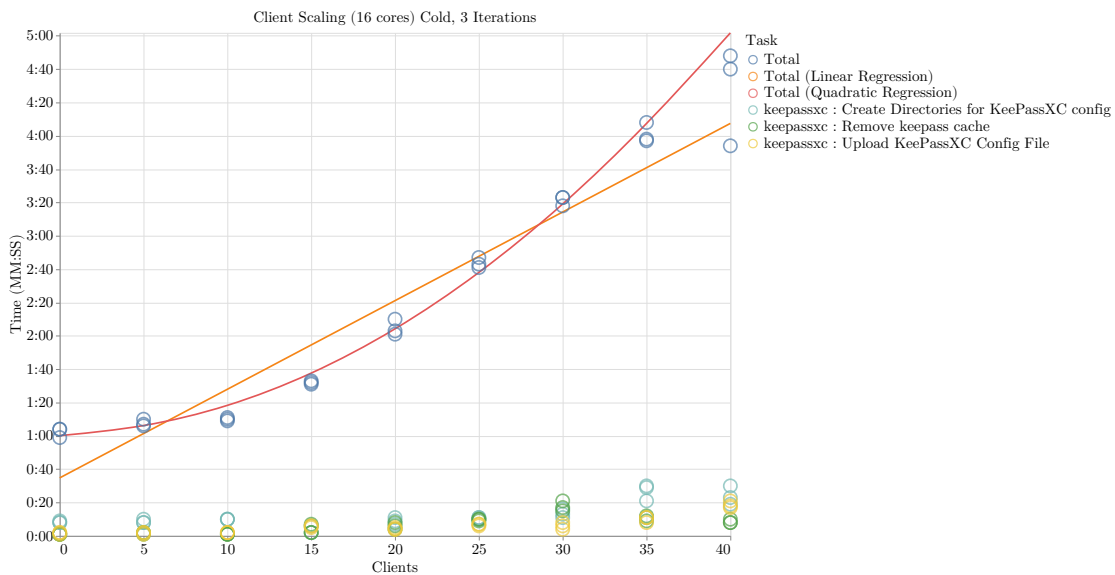


Figure A.5: Client Scaling Cold (Package Cache Component)

A. BENCHMARKS

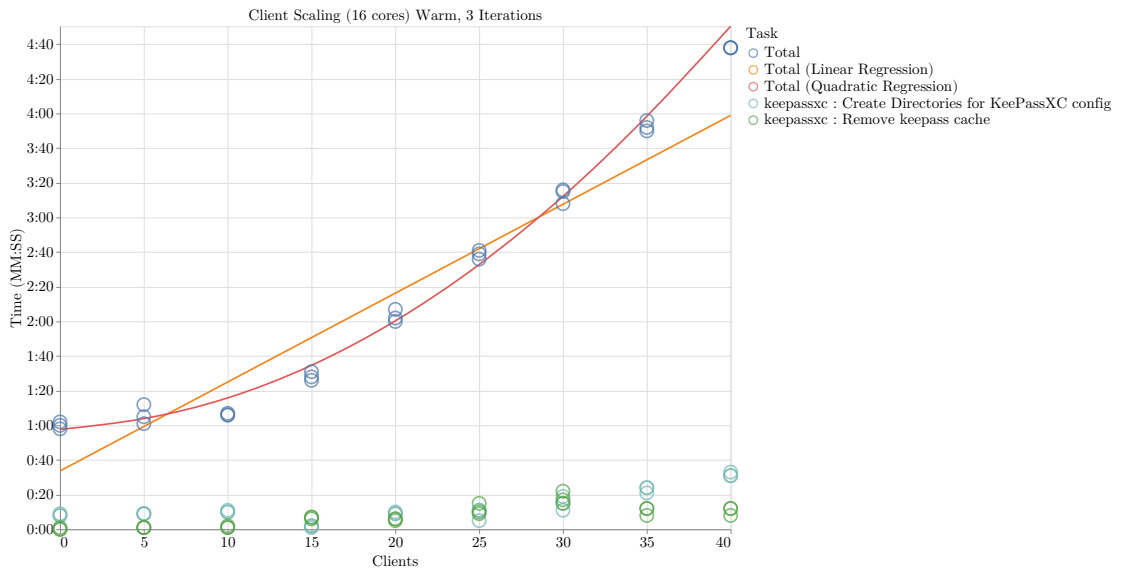


Figure A.6: Client Scaling Warm (Package Cache Component)

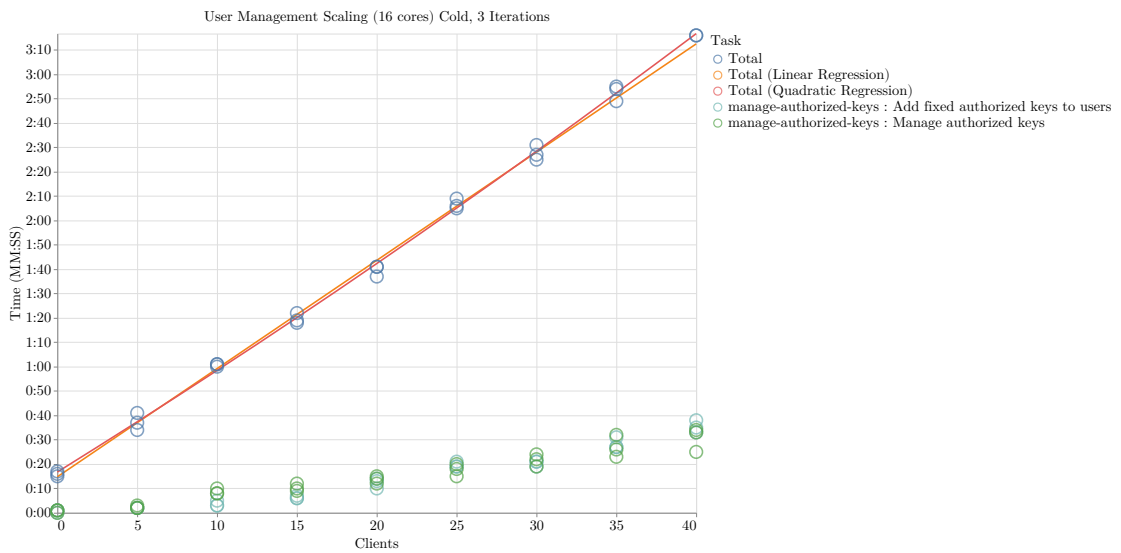


Figure A.7: User Management Scaling Cold (Package Cache Component)

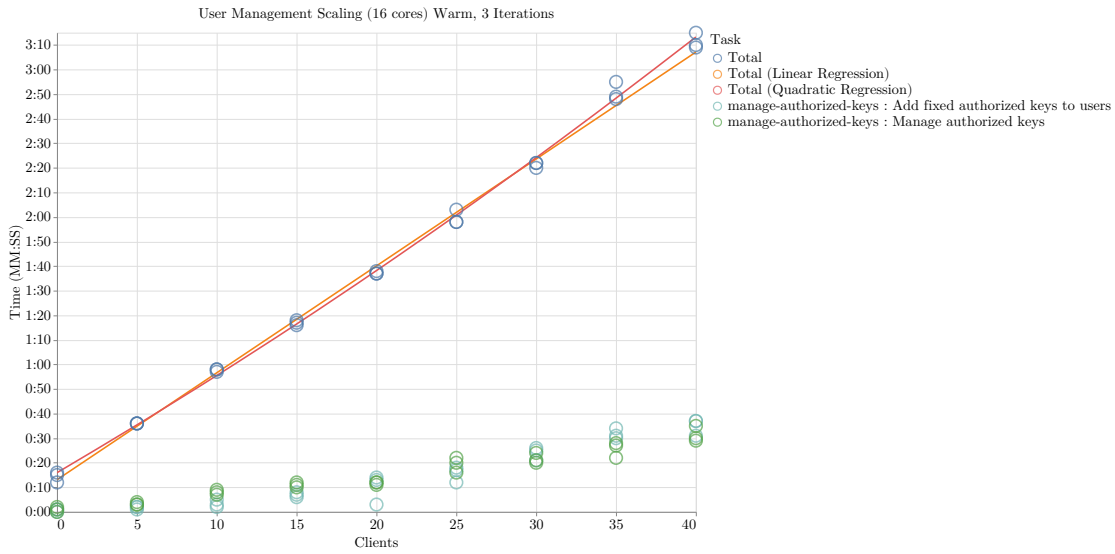


Figure A.8: User Management Scaling Warm (Package Cache Component)

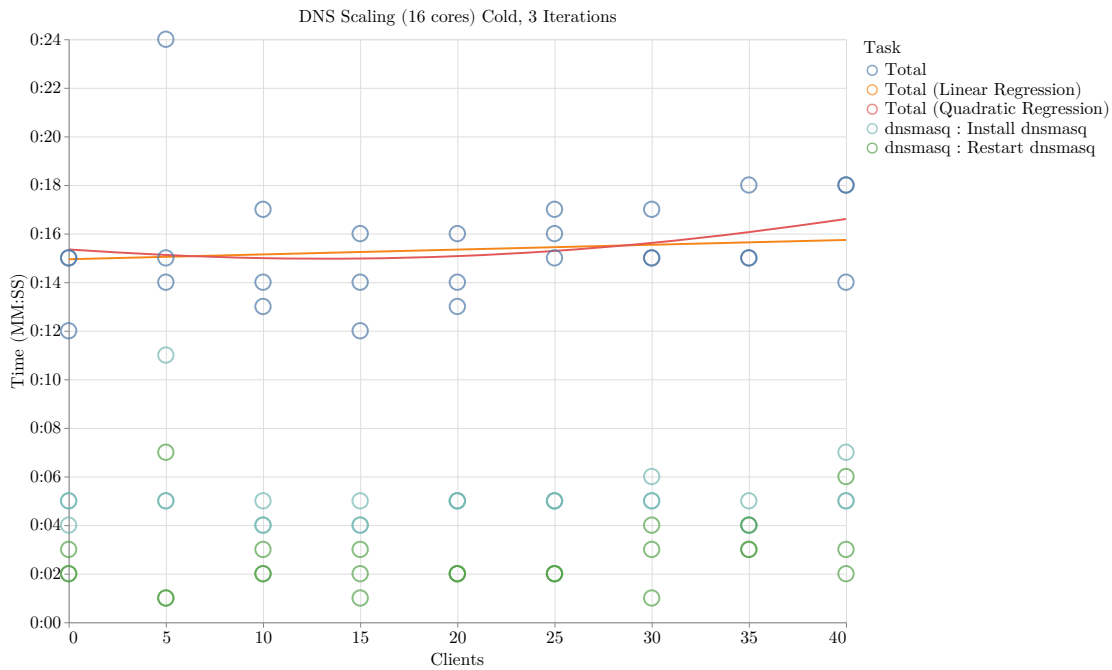


Figure A.9: DNS Scaling Cold (Package Cache Component)

A. BENCHMARKS

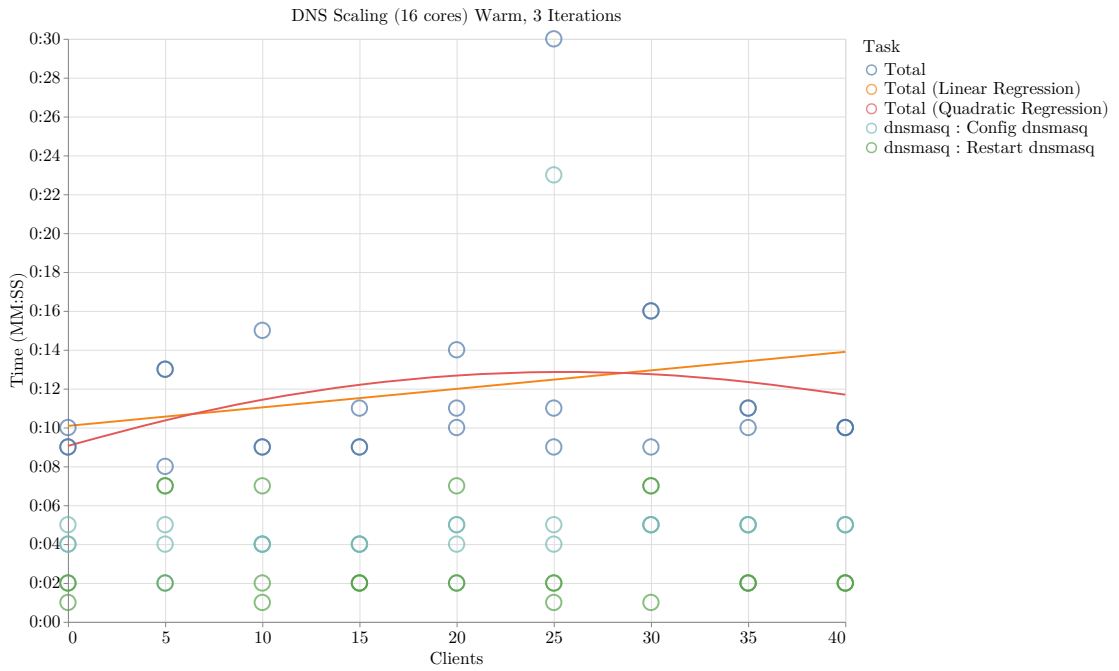


Figure A.10: DNS Scaling Warm (Package Cache Component)

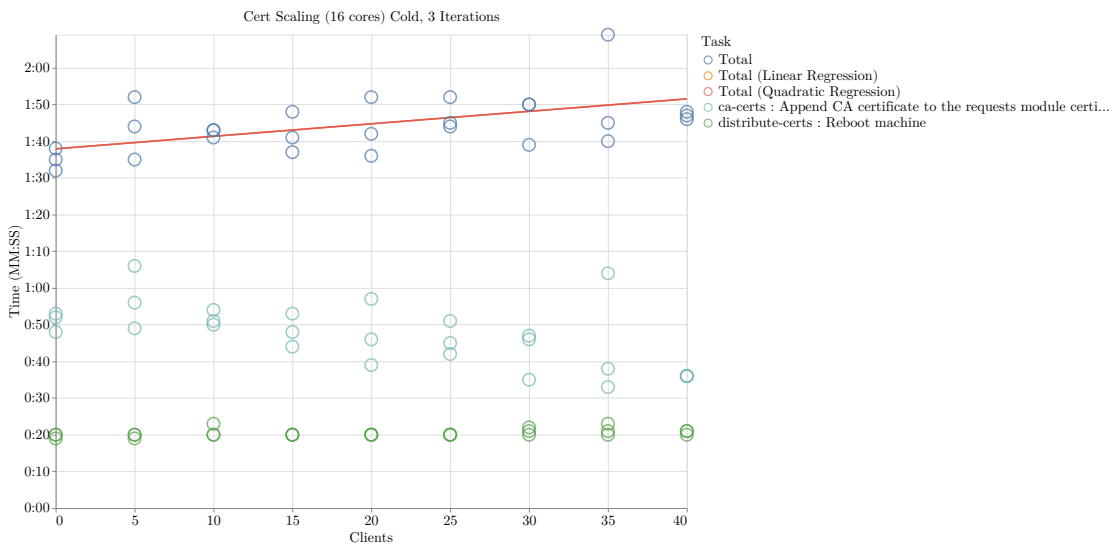


Figure A.11: Certificate Scaling Cold (Package Cache Component)

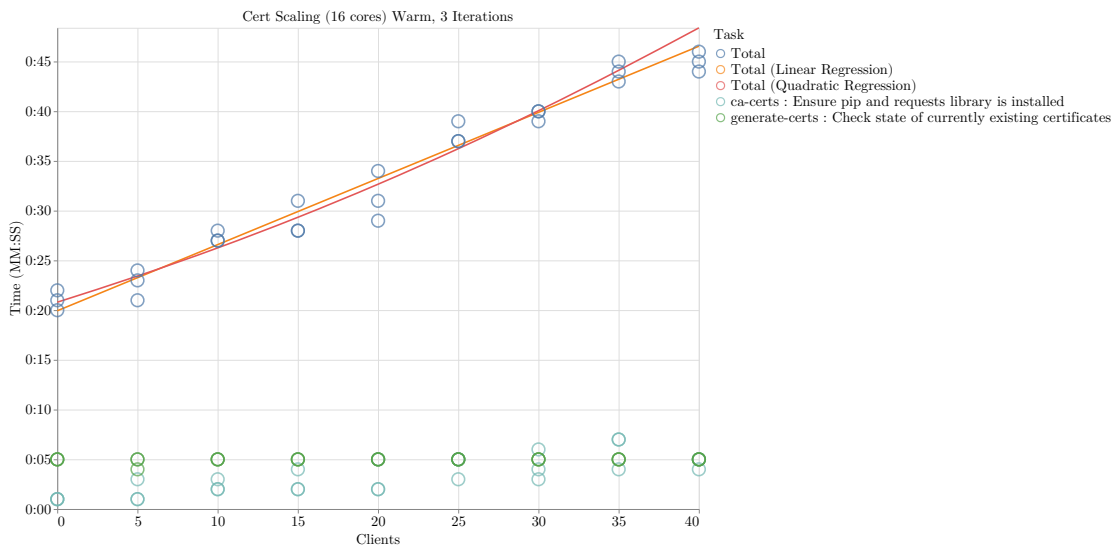


Figure A.12: Certificate Scaling Warm (Package Cache Component)

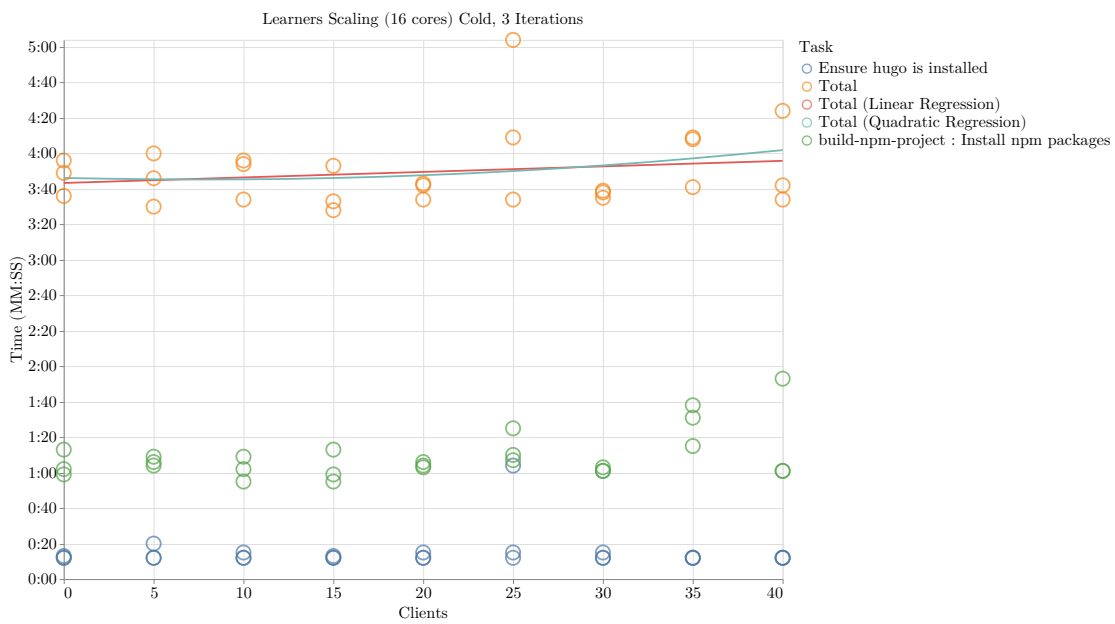


Figure A.13: Learners Scaling Cold (Package Cache Component)

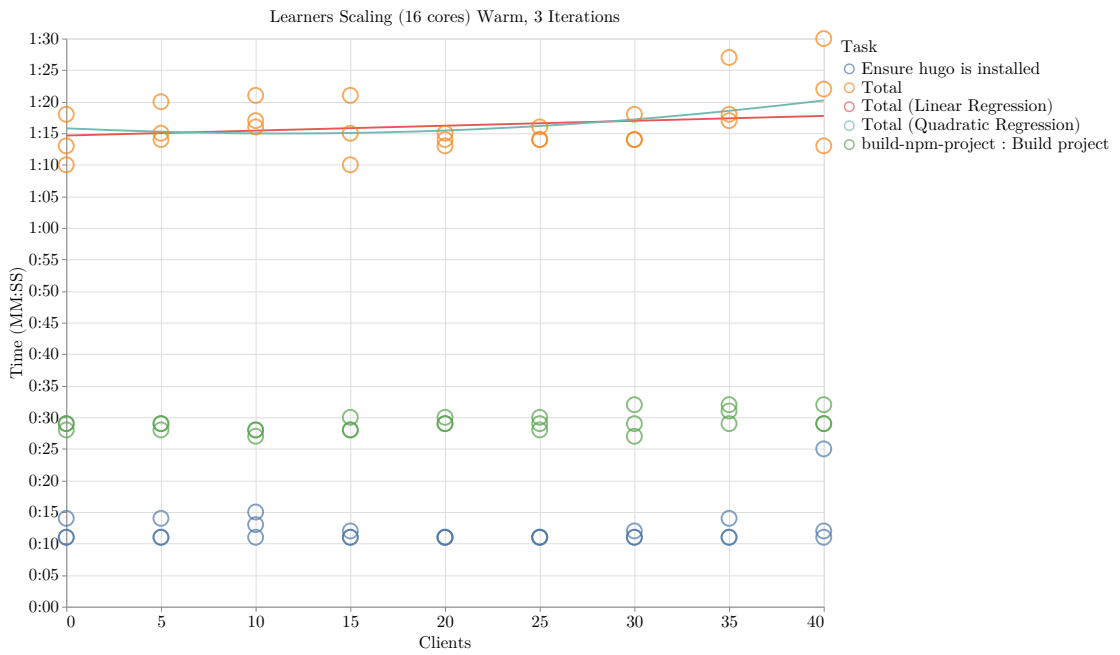


Figure A.14: Learners Scaling Warm (Package Cache Component)

A.3 Multi-Client Component

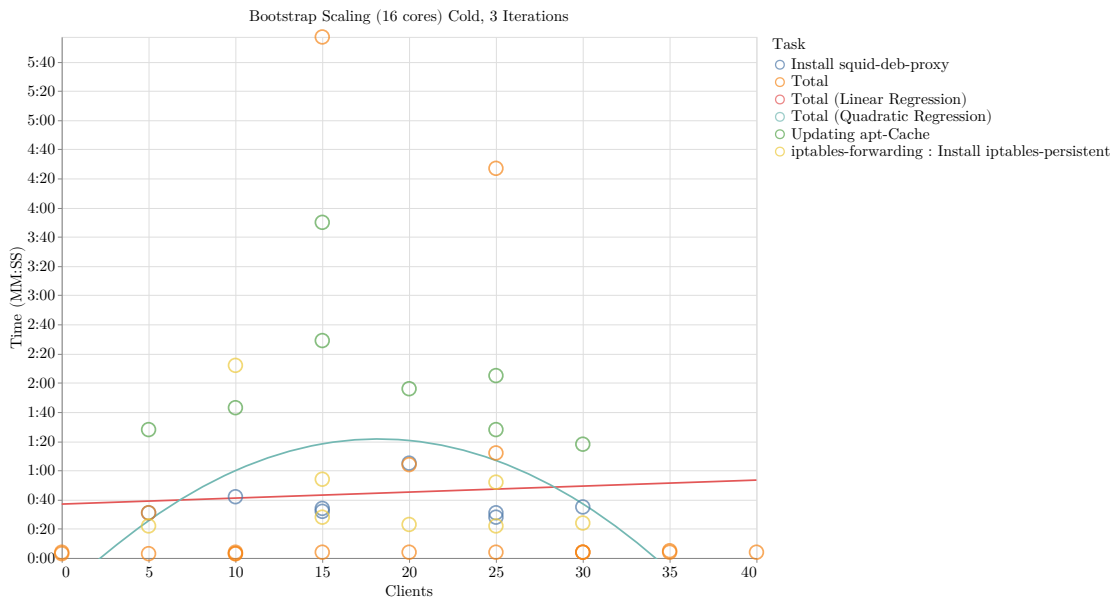


Figure A.15: Bootstrap Scaling Cold (Multi-Client Component)

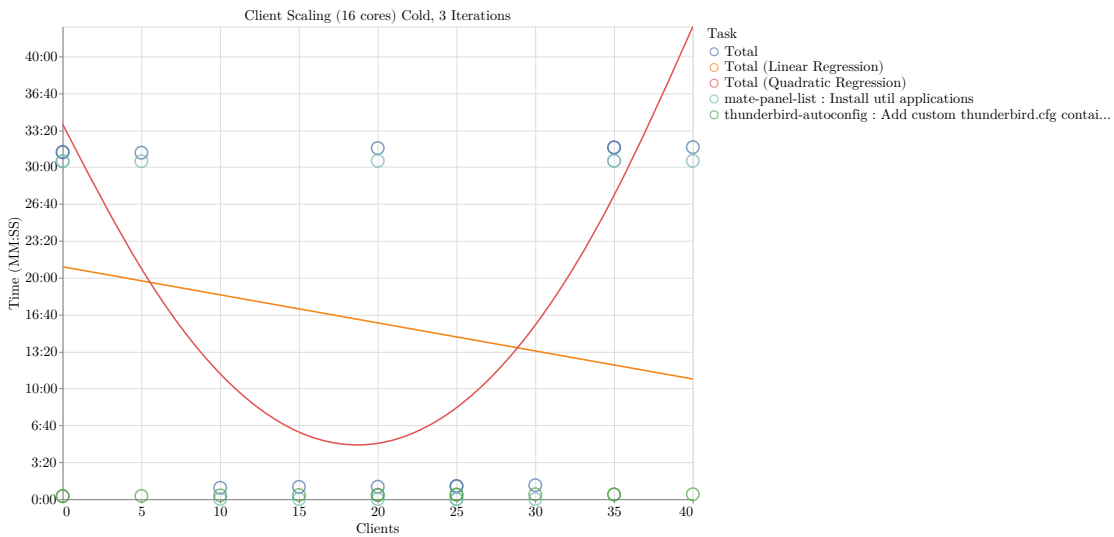


Figure A.16: Client Scaling Cold (Multi-Client Component)

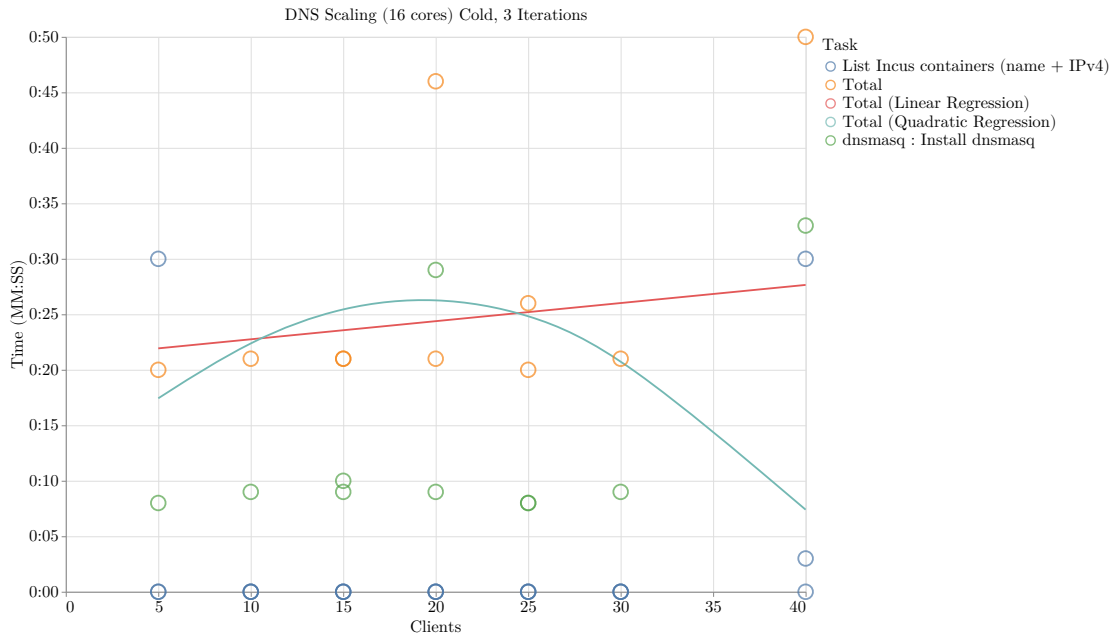


Figure A.17: DNS Scaling Cold (Multi-Client Component)

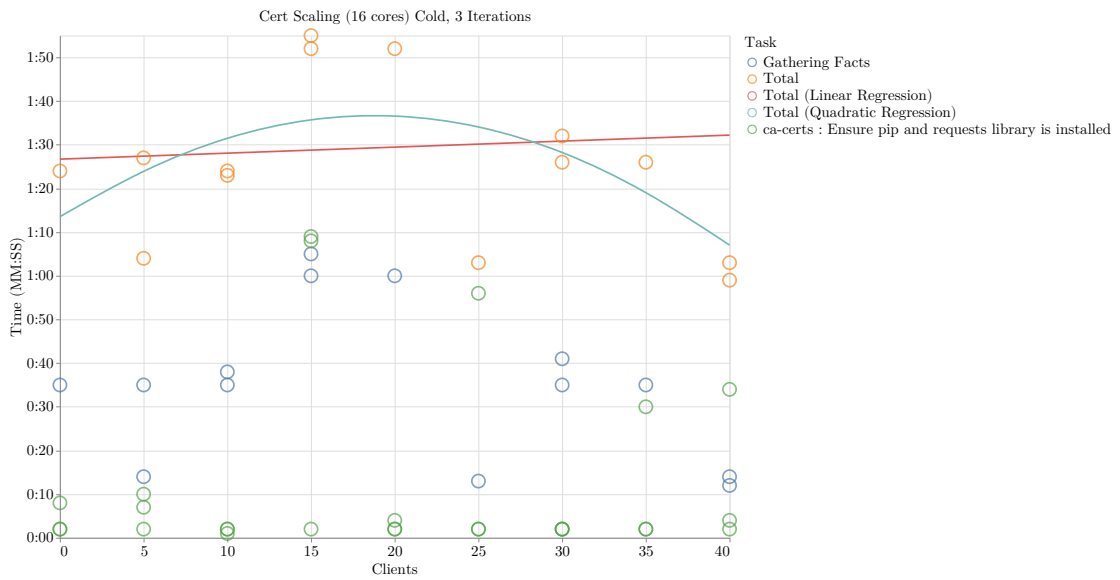


Figure A.18: Certificate Scaling Cold (Multi-Client Component)

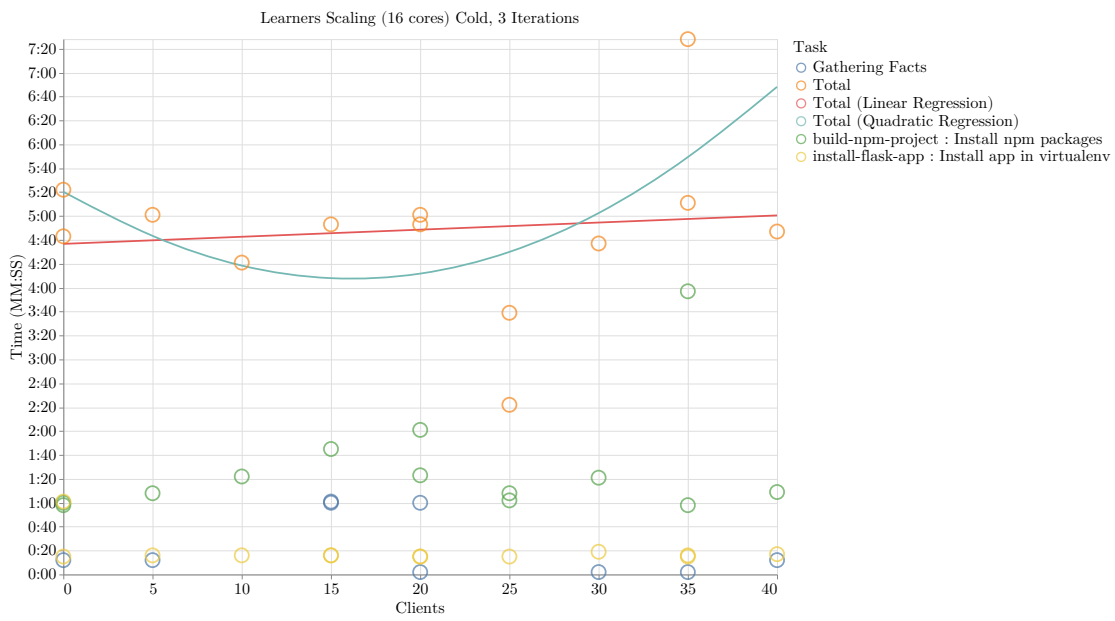


Figure A.19: Learners Scaling Cold (Multi-Client Component)



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

2.1	Example Infrastructure [69]	8
2.2	Cyber Range Exercise Lifecycle	10
3.1	Pattern Selection Activity Diagram	24
3.2	Design and Development Activity Diagram	26
3.3	Infrastructure Deployment Architecture	28
4.1	Bootstrap Scaling Cold	34
4.2	Bootstrap Scaling Warm	34
4.3	Client Scaling Cold	35
4.4	Client Scaling Warm	35
4.5	User Management Scaling Cold	36
4.6	User Management Scaling Warm	36
4.7	DNS Scaling Cold	37
4.8	DNS Scaling Warm	37
4.9	Certificate Scaling Cold	38
4.10	Certificate Scaling Warm	38
4.11	Learners Scaling Cold	39
4.12	Learners Scaling Warm	39
5.1	Radar Diagrams of Metric Estimations per Pattern	49
5.1	Radar Diagrams of Metric Estimations per Pattern	50
5.1	Radar Diagrams of Metric Estimations per Pattern	51
6.1	Bootstrap Scaling Cold (without Proxy Jump Component)	55
6.2	Learners Scaling Cold (without Proxy Jump Component)	55
6.3	Bootstrap Scaling Cold (Package Cache Component)	57
6.4	Bootstrap Scaling Warm (Package Cache Component)	57
6.5	User Management Scaling Cold (Multi-Client Component)	59
A.1	Client Scaling Cold (without Proxy Jump Component)	71
A.2	User Management Scaling Cold (without Proxy Jump Component)	72
A.3	DNS Scaling Cold (without Proxy Jump Component)	72
A.4	Certificate Scaling Cold (without Proxy Jump Component)	73
A.5	Client Scaling Cold (Package Cache Component)	73
		83

A.6 Client Scaling Warm (Package Cache Component)	74
A.7 User Management Scaling Cold (Package Cache Component)	74
A.8 User Management Scaling Warm (Package Cache Component)	75
A.9 DNS Scaling Cold (Package Cache Component)	75
A.10 DNS Scaling Warm (Package Cache Component)	76
A.11 Certificate Scaling Cold (Package Cache Component)	76
A.12 Certificate Scaling Warm (Package Cache Component)	77
A.13 Learners Scaling Cold (Package Cache Component)	77
A.14 Learners Scaling Warm (Package Cache Component)	78
A.15 Bootstrap Scaling Cold (Multi-Client Component)	79
A.16 Client Scaling Cold (Multi-Client Component)	79
A.17 DNS Scaling Cold (Multi-Client Component)	80
A.18 Certificate Scaling Cold (Multi-Client Component)	80
A.19 Learners Scaling Cold (Multi-Client Component)	81

List of Tables

5.1	Pattern Relevance	47
5.2	Pattern Metric Expert	49



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

- CoW** Copy on Write. 64
- CSIRT** Computer Security Incident Response Team. 8
- DMZ** Demilitarized Zone. 7
- DORA** Digital Operational Resilience Act. 1
- DSL** Domain Specific Language. 18
- GDPR** General Data Protection Regulation. 9
- HCL** HashiCorp Configuration Language. 18
- IaaS** Infrastructure as a Service. 13, 15
- IaC** Infrastructure as Code. 2, 11, 13, 16, 17, 28
- IT** Information Technology. 7
- KOF** Kick-out Factor. xi, xiii, 4, 45
- KPI** Key Performance Indicator. 26, 27
- NIS** Network and Information Systems. 1, 9
- OT** Operational Technology. 7
- SLA** Service Level Agreement. 15



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [1] *5 Things Toxic to Scalability - iheavy - Devops + Cloud Solutions Architect* — *iheavy.com*. <https://www.iheavy.com/5-things-toxic-to-scalability/>. [Accessed 09-10-2025].
- [2] Martin L. Abbott and Michael T. Fisher. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*. en. Google-Books-ID: yzUpD2YbhWwC. Pearson Education, Dec. 2009. ISBN: 978-0-13-703139-9.
- [3] *Amazon EC2 - Cloud Compute Capacity - AWS* — *aws.amazon.com*. <http://aws.amazon.com/ec2/>. [Accessed 09-10-2025].
- [4] Tanenbaum S Andrew and Maarten Van Steen. „Distributed systems-principles and paradigms, 2nd Edition“. In: (2007).
- [5] *Ansible AWX community documentation*. <https://ansible.readthedocs.io/projects/awx/en/latest/>. [Accessed 15-09-2025].
- [6] *Ansible Collaborative* — *ansible.com*. www.ansible.com. [Accessed 19-03-2025].
- [7] Claudio A. Ardagna et al. „Scalability Patterns for Platform-as-a-Service“. In: *2012 IEEE Fifth International Conference on Cloud Computing*. ISSN: 2159-6190. June 2012, pp. 718–725. DOI: 10.1109/CLOUD.2012.41. URL: <https://ieeexplore.ieee.org/abstract/document/6253571> (visited on 04/10/2025).
- [8] D. Armstrong and K. Djemame. „Performance Issues in Clouds: An Evaluation of Virtual Image Propagation and I/O Paravirtualization“. en. In: *The Computer Journal* 54.6 (June 2011), pp. 836–849. ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/bxr011. URL: <https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/bxr011> (visited on 02/17/2025).
- [9] Asia Pacific University of Technology & Innovation, Malaysia & Universiti Putra Malaysia. et al. „Scalability Enhancement for Cloud-based Applications using Software Oriented Methods“. In: *International Journal of Engineering and Advanced Technology* 8.6 (Aug. 2019), pp. 4208–4213. ISSN: 22498958. DOI: 10.35940/ijeat.F8869.088619. URL: <https://www.ijeat.org/portfolio-item/F8869088619/> (visited on 07/03/2025).

- [10] Alberto Avritzer et al. „Scalability Assessment of Microservice Architecture Deployment Configurations: A Domain-based Approach Leveraging Operational Profiles and Load Tests“. In: *Journal of Systems and Software* 165 (July 2020), p. 110564. ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110564. URL: <https://www.sciencedirect.com/science/article/pii/S016412122030042X> (visited on 07/03/2025).
- [11] *Azure Resource Manager Templates | Microsoft Azure* — [azure.microsoft.com](https://azure.microsoft.com/en-us/products/arm-templates). <https://azure.microsoft.com/en-us/products/arm-templates>. [Accessed 19-03-2025].
- [12] Amine Barkat, Alysso Diniz Dos Santos, and Thi Thao Nguyen Ho. „Open Stack and Cloud Stack: Open Source Solutions for Building Public and Private Clouds“. In: *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Timisoara: IEEE, Sept. 2014, pp. 429–436. ISBN: 978-1-4799-8447-3 978-1-4799-8448-0. DOI: 10.1109/SYNASC.2014.64. URL: <https://ieeexplore.ieee.org/document/7034714/> (visited on 02/17/2025).
- [13] Thomas Bartz-Beielstein et al. *Benchmarking in Optimization: Best Practice and Open Issues*. arXiv:2007.03488 [cs]. Dec. 2020. DOI: 10.48550/arXiv.2007.03488. URL: <http://arxiv.org/abs/2007.03488> (visited on 04/29/2025).
- [14] Philip A. Bernstein and Eric Newcomer. *Principles of Transaction Processing*. en. Google-Books-ID: LmHgK5KKrQQC. Morgan Kaufmann, July 2009. ISBN: 978-0-08-094841-6.
- [15] André B. Bondi. „Characteristics of scalability and their impact on performance“. en. In: *Proceedings of the 2nd international workshop on Software and performance*. Ottawa Ontario Canada: ACM, Sept. 2000, pp. 195–203. ISBN: 978-1-58113-195-6. DOI: 10.1145/350391.350432. URL: <https://dl.acm.org/doi/10.1145/350391.350432> (visited on 02/24/2025).
- [16] *Bugs in package apt-cacher-ng – Debian Bug report logs* — [bugs.debian.org](https://bugs.debian.org/cgi-bin/pkgreport.cgi?pkg=apt-cacher-ng). <https://bugs.debian.org/cgi-bin/pkgreport.cgi?pkg=apt-cacher-ng>. [Accessed 06-11-2025].
- [17] David A. Chappell. *Enterprise Service Bus: Theory in Practice*. en. Google-Books-ID: Uhue3faV2mwC. "O'Reilly Media, Inc.", June 2004. ISBN: 978-1-4493-9109-6.
- [18] *Chef Software DevOps Automation Solutions | Chef* — [chef.io](https://www.chef.io). <https://www.chef.io>. [Accessed 19-03-2025].
- [19] *Cloud Computing Services - Amazon Web Services (AWS)* — aws.amazon.com. <https://aws.amazon.com>. [Accessed 19-03-2025].
- [20] *Cloud Computing Services | Google Cloud* — cloud.google.com. <https://cloud.google.com>. [Accessed 19-03-2025].
- [21] *Cloud Computing Services | Microsoft Azure* — azure.microsoft.com. <https://azure.microsoft.com>. [Accessed 19-03-2025].

- [22] Bruce Powel Douglass. *Real-time Design Patterns: Robust Scalable Architecture for Real-time Systems*. en. Google-Books-ID: drlsKjcw3xQC. Addison-Wesley Professional, 2003. ISBN: 978-0-201-69956-2.
- [23] Christoph Fehling et al. „An architectural pattern language of cloud-based applications“. In: *Proceedings of the 18th Conference on Pattern Languages of Programs*. PLoP '11. New York, NY, USA: Association for Computing Machinery, Oct. 2011, pp. 1–11. ISBN: 978-1-4503-1283-7. DOI: 10.1145/2578903.2579140. URL: <https://dl.acm.org/doi/10.1145/2578903.2579140> (visited on 07/03/2025).
- [24] Christoph Fehling et al. „Capturing Cloud Computing Knowledge and Experience in Patterns“. In: *2012 IEEE Fifth International Conference on Cloud Computing*. ISSN: 2159-6190. June 2012, pp. 726–733. DOI: 10.1109/CLOUD.2012.124. URL: <https://ieeexplore.ieee.org/abstract/document/6253572> (visited on 10/09/2025).
- [25] Christoph Fehling et al. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. en. Vienna: Springer Vienna, 2014. ISBN: 978-3-7091-1567-1 978-3-7091-1568-8. DOI: 10.1007/978-3-7091-1568-8. URL: <https://link.springer.com/10.1007/978-3-7091-1568-8> (visited on 02/17/2025).
- [26] Christoph Fehling et al. „Pattern-Based Development and Management of Cloud Applications“. en. In: *Future Internet* 4.1 (Feb. 2012), pp. 110–141. ISSN: 1999-5903. DOI: 10.3390/fi4010110. URL: <https://www.mdpi.com/1999-5903/4/1/110> (visited on 09/29/2025).
- [27] Roy T. Fielding and Richard N. Taylor. „Principled design of the modern Web architecture“. In: *ACM Trans. Internet Technol.* 2.2 (May 2002), pp. 115–150. ISSN: 1533-5399. DOI: 10.1145/514183.514185. URL: <https://dl.acm.org/doi/10.1145/514183.514185> (visited on 09/29/2025).
- [28] M. Fowler. „Data access routines“. In: *IEEE Software* 20.6 (Nov. 2003), pp. 96–98. ISSN: 1937-4194. DOI: 10.1109/MS.2003.1241375. URL: <https://ieeexplore.ieee.org/abstract/document/1241375> (visited on 10/09/2025).
- [29] Martin Fowler. *Patterns of Enterprise Application Architecture*. en. Google-Books-ID: vqTfNFDzdzIC. Addison-Wesley, Mar. 2012. ISBN: 978-0-13-306521-3.
- [30] Katharina Gilles et al. „Proteus Hypervisor: Full Virtualization and Paravirtualization for Multi-core Embedded Systems“. en. In: *Embedded Systems: Design, Analysis and Verification*. Ed. by Gunar Schirner et al. Berlin, Heidelberg: Springer, 2013, pp. 293–305. ISBN: 978-3-642-38853-8. DOI: 10.1007/978-3-642-38853-8_27.
- [31] *GitHub - ahang/apt-cacher-ng* — [github.com](https://github.com/ahang/apt-cacher-ng). <https://github.com/ahang/apt-cacher-ng>. [Accessed 01-08-2025].
- [32] *GitHub - mvo5/squid-deb-proxy: Squid proxy configuration to optimize package downloads* — [github.com](https://github.com/mvo5/squid-deb-proxy). <https://github.com/mvo5/squid-deb-proxy>. [Accessed 01-08-2025].

- [33] *Google Cloud Deployment Manager documentation | Cloud Deployment Manager Documentation — cloud.google.com.* <https://cloud.google.com/deployment-manager/docs>. [Accessed 19-03-2025].
- [34] Michele Guerriero et al. „Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry“. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Cleveland, OH, USA: IEEE, Sept. 2019, pp. 580–589. ISBN: 978-1-7281-3094-1. DOI: 10.1109/ICSME.2019.00092. URL: <https://ieeexplore.ieee.org/document/8919181/> (visited on 02/19/2025).
- [35] Rui Han et al. „Lightweight Resource Scaling for Cloud Applications“. In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. May 2012, pp. 644–651. DOI: 10.1109/CCGrid.2012.52. URL: <https://ieeexplore.ieee.org/abstract/document/6217477> (visited on 07/03/2025).
- [36] Mark Hapner et al. „Java message service“. In: *Sun Microsystems Inc., Santa Clara, CA 9* (2002).
- [37] Lena Hatzelhoffer and Michael Lobeck. *Smart City in Practice: Innovation Lab Between Vision and Reality*. Jovis, 2012.
- [38] Sören Henning and Wilhelm Hasselbring. „A configurable method for benchmarking scalability of cloud-native applications“. en. In: *Empirical Software Engineering* 27.6 (Aug. 2022), p. 143. ISSN: 1573-7616. DOI: 10.1007/s10664-022-10162-1. URL: <https://doi.org/10.1007/s10664-022-10162-1> (visited on 04/10/2025).
- [39] T Hoff. *Strategy: Guaranteed Availability Requires Reserving Instances in Specific Zones*. <http://highscalability.com/blog/2011/12/28/strategy-guaranteed-availability-requires-reserving-instance.html>. [Accessed 09-10-2025].
- [40] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. en. Google-Books-ID: bUlsAQAAQBAJ. Addison-Wesley Professional, 2004. ISBN: 978-0-321-20068-6.
- [41] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. en. Google-Books-ID: bUlsAQAAQBAJ. Addison-Wesley Professional, 2004. ISBN: 978-0-321-20068-6.
- [42] *HOME — asc-s.de*. <http://www.asc-s.de/>. [Accessed 09-10-2025].
- [43] Michael Howard. *Terraform – Automating Infrastructure as a Service*. Version Number: 1. 2022. DOI: 10.48550/ARXIV.2205.10676. URL: <https://arxiv.org/abs/2205.10676> (visited on 02/19/2025).
- [44] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. en. Pearson Education, July 2010. ISBN: 978-0-321-67022-9.

- [45] *Information service patterns, part 4: Master data management architecture patterns*. <https://web.archive.org/web/20170109075956/http://www.ibm.com/developerworks/data/library/techarticle/dm-0703sauter/>. [Accessed 29-09-2025].
- [46] Vyron Kampourakis, Vasileios Gkioulos, and Sokratis Katsikas. „A step-by-step definition of a reference architecture for cyber ranges“. In: *Journal of Information Security and Applications* 88 (Feb. 2025), p. 103917. ISSN: 2214-2126. DOI: 10.1016/j.jisa.2024.103917. URL: <https://www.sciencedirect.com/science/article/pii/S2214212624002199> (visited on 08/04/2025).
- [47] Murat Karakus and Arjan Duresi. „A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)“. eng. In: *Computer networks (Amsterdam, Netherlands : 1999)* 112 (2017), pp. 279–293. ISSN: 1389-1286.
- [48] Murat Karakus and Arjan Duresi. „A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)“. en. In: *Computer Networks* 112 (Jan. 2017), pp. 279–293. ISSN: 13891286. DOI: 10.1016/j.comnet.2016.11.017. URL: <https://linkinghub.elsevier.com/retrieve/pii/S138912861630411X> (visited on 02/17/2025).
- [49] Rajesh Komar and Arjun Patil. „Emerging Trends in Cloud Computing: A Comprehensive Analysis of Deployment Models and Service Models for Scalability, Flexibility, and Security Enhancements.“ en. In: *Journal of Intelligent Systems and Applied Data Science* 1.1 (July 2023). ISSN: 2974-9840. URL: <https://jisads.com/index.php/1/article/view/10> (visited on 07/03/2025).
- [50] Jukka Koskelin. „Modular Infrastructure as Code in Azure PaaS“. In: (2023). Publisher: Master’s thesis, University of Helsinki.
- [51] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-oriented Architecture Best Practices*. en. Google-Books-ID: R7oGhITYUuUC. Prentice Hall Professional, 2005. ISBN: 978-0-13-146575-6.
- [52] Karel Kuchar, Petr Blazek, and Radek Fujdiak. „From Playground to Battleground: Cyber Range Training for Industrial Cybersecurity Education“. eng. In: *ACM International Conference Proceeding Series*. New York, NY, USA: ACM, 2023, pp. 209–214. ISBN: 9798400707964.
- [53] Indika Kumara et al. „The do’s and don’ts of infrastructure code: A systematic gray literature review“. en. In: *Information and Software Technology* 137 (Sept. 2021), p. 106593. ISSN: 09505849. DOI: 10.1016/j.infsof.2021.106593. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0950584921000720> (visited on 02/17/2025).
- [54] Maria Leitner et al. „AIT Cyber Range: Flexible Cyber Security Environment for Exercises, Training and Research“. en. In: *Proceedings of the European Interdisciplinary Cybersecurity Conference*. Rennes France: ACM, Nov. 2020, pp. 1–6. ISBN: 978-1-4503-7599-3. DOI: 10.1145/3424954.3424959. URL: <https://dl.acm.org/doi/10.1145/3424954.3424959> (visited on 02/17/2025).

- [55] Maria Leitner et al. „AIT Cyber Range: Flexible Cyber Security Environment for Exercises, Training and Research“. In: *Proceedings of the 2020 European Interdisciplinary Cybersecurity Conference*. EICC '20. Rennes, France: Association for Computing Machinery, 2021. ISBN: 9781450375993. DOI: 10.1145/3424954.3424959. URL: <https://doi.org/10.1145/3424954.3424959>.
- [56] Frank Leymann and Dieter Roller. *Production Workflow - Concepts and Techniques*. Englisch. PTR Prentice Hall, Jan. 2000, p. 479. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=BOOK-2000-01&engl=0.
- [57] Kari K. Lilja, Kimmo Laakso, and Jari Palomäki. „Using the Delphi method“. In: *2011 Proceedings of PICMET '11: Technology Management in the Energy Smart World (PICMET)*. ISSN: 2159-5100. July 2011, pp. 1–10. URL: <https://ieeexplore.ieee.org/abstract/document/6017716> (visited on 09/18/2025).
- [58] *Linux Containers - Incus - Introduction* — linuxcontainers.org. <https://linuxcontainers.org/incus/>. [Accessed 10-10-2025].
- [59] Pavel Masek et al. „Unleashing Full Potential of Ansible Framework: University Labs Administration“. In: *2018 22nd Conference of Open Innovations Association (FRUCT)*. Jyvaskyla: IEEE, May 2018, pp. 144–150. ISBN: 978-952-68653-4-8. DOI: 10.23919/FRUCT.2018.8468270. URL: <https://ieeexplore.ieee.org/document/8468270/> (visited on 02/17/2025).
- [60] *NIS Directive* — [enisa.europa.eu](https://www.enisa.europa.eu). <https://www.enisa.europa.eu/topics/cybersecurity-policy/nis-directive-new>. [Accessed 18-09-2024].
- [61] *Open Source Cloud Computing Infrastructure - OpenStack* — [openstack.org](https://www.openstack.org). <https://www.openstack.org>. [Accessed 19-03-2025].
- [62] *OpenTofu* — opentofu.org. <https://opentofu.org>. [Accessed 19-03-2025].
- [63] Pradeep Padala et al. „Automated control of multiple virtualized resources“. In: *Proceedings of the 4th ACM European conference on Computer systems*. EuroSys '09. New York, NY, USA: Association for Computing Machinery, Apr. 2009, pp. 13–26. ISBN: 978-1-60558-482-9. DOI: 10.1145/1519065.1519068. URL: <https://dl.acm.org/doi/10.1145/1519065.1519068> (visited on 02/24/2025).
- [64] R. S. M. Lakshmi Patibandla, Santhi Sri Kurra, and Nirupama Bhat Mundukur. „A Study on Scalability of Services and Privacy Issues in Cloud Computing“. In: *Distributed Computing and Internet Technology*. Ed. by R. Ramanujam and Srini Ramaswamy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 212–230. ISBN: 978-3-642-28073-3.
- [65] *Pulumi - Infrastructure as Code, Secrets Management, and AI* — [pulumi.com](https://www.pulumi.com). <https://www.pulumi.com>. [Accessed 19-03-2025].
- [66] *Puppet Infrastructure & IT Automation at Scale | Puppet by Perforce* — [puppet.com](https://www.puppet.com). <https://www.puppet.com>. [Accessed 19-03-2025].

- [67] *Red Hat Ansible Automation Platform* — *redhat.com*. <https://www.redhat.com/en/technologies/management/ansible>. [Accessed 15-09-2025].
- [68] *Regulation - 2022/2554 - EN - DORA - EUR-Lex* — *eur-lex.europa.eu*. <https://eur-lex.europa.eu/eli/reg/2022/2554/oj?uri=CELEX:32022R2554>. [Accessed 18-09-2024].
- [69] Lenhard Reuter. „Infrastructure of the AIT Cyber Range“. Used for a presentation but never has been published in a paper or thesis.
- [70] Tiago Rosado and Jorge Bernardino. „An overview of openstack architecture“. en. In: *Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS '14*. Porto, Portugal: ACM Press, 2014, pp. 366–367. ISBN: 978-1-4503-2627-8. DOI: 10.1145/2628194.2628195. URL: <http://dl.acm.org/citation.cfm?doid=2628194.2628195> (visited on 02/17/2025).
- [71] *S3 FAQs All* — *aws.amazon.com*. <https://aws.amazon.com/s3/faqs/>. [Accessed 09-10-2025].
- [72] *Scalr | Features* — *scalr.net*. <http://scalr.net/features/>. [Accessed 09-10-2025].
- [73] *Secure Storage - Amazon S3 Glacier storage classes - AWS* — *aws.amazon.com*. <http://aws.amazon.com/glacier/>. [Accessed 09-10-2025].
- [74] Ensar Seker and Hasan Huseyin Ozbenli. „The Concept of Cyber Defence Exercises (CDX): Planning, Execution, Evaluation“. In: *2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. June 2018, pp. 1–9. DOI: 10.1109/CyberSecPODS.2018.8560673.
- [75] Nicolás Serrano, Gorika Gallardo, and Josune Hernantes. „Infrastructure as a Service and Cloud Technologies“. In: *IEEE Software* 32.2 (Mar. 2015), pp. 30–36. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2015.43. URL: <https://ieeexplore.ieee.org/document/7057553/> (visited on 02/17/2025).
- [76] Daniel Sokolowski. „Infrastructure as code for dynamic deployments“. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 1775–1779. ISBN: 978-1-4503-9413-0. DOI: 10.1145/3540250.3558912. URL: <https://dl.acm.org/doi/10.1145/3540250.3558912> (visited on 04/10/2025).
- [77] *Spot Instance - Amazon EC2 Spot Instances - AWS* — *aws.amazon.com*. <https://aws.amazon.com/ec2/spot/>. [Accessed 09-10-2025].
- [78] *squid : Optimising Web Delivery* — *squid-cache.org*. <https://www.squid-cache.org/>. [Accessed 01-08-2025].
- [79] *ssh(1) General Commands Manual*. Dec. 2024.
- [80] Brent Stephens et al. „A Scalability Study of Enterprise Network Architectures“. In: *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*. 2011, pp. 111–121. DOI: 10.1109/ANCS.2011.28.

- [81] *Terraform by HashiCorp — terraform.io*. <https://www.terraform.io>. [Accessed 19-03-2025].
- [82] Jinesh Varia. „Architecting for the cloud: Best practices“. In: (2010).
- [83] Jinesh Varia. „Cloud architectures“. In: *White Paper of Amazon, jineshvaria. s3.amazonaws.com/public/cloudarchitectures-varia.pdf* 16 (2008).
- [84] Jan Vykopal et al. „Lessons learned from complex hands-on defence exercises in a cyber range“. In: *2017 IEEE Frontiers in Education Conference (FIE)*. 2017, pp. 1–8. DOI: 10.1109/FIE.2017.8190713.
- [85] *Web App Deployment - AWS Elastic Beanstalk - AWS — aws.amazon.com*. <http://aws.amazon.com/elasticbeanstalk/>. [Accessed 09-10-2025].
- [86] Sanjiva Weerawarana et al. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. USA: Prentice Hall PTR, Feb. 2005. ISBN: 978-0-13-148874-8.
- [87] Duane Wessels. *Web Caching*. en. Google-Books-ID: fVuWayXLdYIC. "O'Reilly Media, Inc.", 2001. ISBN: 978-1-56592-536-6.
- [88] *What is AWS CloudFormation? - AWS CloudFormation — docs.aws.amazon.com*. <https://docs.aws.amazon.com/cloudformation>. [Accessed 19-03-2025].
- [89] wibjorn. *Architecture strategies for catching the long tail*. <https://web.archive.org/web/20090218023802/http://msdn.microsoft.com/en-us/library/aa479069.aspx>. [Accessed 29-09-2025].
- [90] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. en. Berlin, Heidelberg: Springer, 2014. ISBN: 978-3-662-43838-1 978-3-662-43839-8. DOI: 10.1007/978-3-662-43839-8. URL: <https://link.springer.com/10.1007/978-3-662-43839-8> (visited on 06/17/2025).
- [91] Muhammad Mudassar Yamin, Basel Katt, and Vasileios Gkioulos. „Cyber ranges and security testbeds: Scenarios, functions, tools and architecture“. en. In: *Computers & Security* 88 (Jan. 2020), p. 101636. ISSN: 01674048. DOI: 10.1016/j.cose.2019.101636. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0167404819301804> (visited on 02/17/2025).