Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Karishma Sharma
Mathias Westby Skoglund
Vegard Færgestad

# Streamlining the Provisioning of Physical Clients: A Secure Approach for an Environment with Low Risk Acceptance

Bachelor's thesis in Computer Science
Supervisor: Kiran Raja

May 2023

NTNU
Norwegian University of
Science and Technology

Karishma Sharma
Mathias Westby Skoglund
Vegard Færgestad

# Streamlining the Provisioning of Physical Clients: A Secure Approach for an Environment with Low Risk Acceptance

Bachelor's thesis in Computer Science
Supervisor: Kiran Raja
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

# Abstract

There is a need for an automatic way of provisioning physical clients in secure environments, as the solutions already existing lacks the necessary functionality. In this project, a solution for the automatic provisioning of physical clients was developed, encompassing installation and configuration in a low risk acceptance environment. The objective was to minimize manual actions, ensuring that a client transitions from factory settings to a fully configured state without increasing the risk in the deployment environment. Prior to determining a potential solution architecture in collaboration with the employer, research was conducted on existing alternatives. The development process employed agile methodologies, featuring short sprints and frequent employer feedback. A security assessment of the solution was conducted using the employer's framework, and both the solution and security assessment were evaluated by experts at the employer's organization. User testing was also conducted.

The solution consists of a network boot server utilizing UEFI HTTP BOOT, a DNS server for domain name resolution, a DHCP server for network access and boot options, a self-developed orchestration service to manage the process, and an Ansible Automation Platform instance for client configuration post-OS installation. The security assessment indicated that the solution maintains an acceptably low risk level. Expert evaluations confirmed that both the solution and security assessment satisfy the employer's requirements.

User testing, expert evaluation, and personal experience form the foundation for future work, which should prioritize the implementation of additional security measures, a security assessment tailored to each environment, and testing the provisioning of multiple clients simultaneously.

# Sammendrag

Det er et behov for en automatisk løsning for å tanke fysiske klienter i sikre miljøer, siden de eksisterende løsningene mangler den nødvendige funksjonaliteten. I dette prosjektet ble det utviklet en løsning for automatisk tanking av fysiske klienter, som omfatter installasjon og konfigurasjon i et miljø med lav risikotoleranse. Målet var å minimere manuelle handlinger, og sikre at en klient går fra fabrikkinnstillinger til en fullt konfigurert tilstand uten å øke risikoen i IT-miljøet. Før en potensiell arkitektur ble bestemt i samarbeid med oppdragsgiver, ble det gjort undersøkelser av eksisterende alternativer. Under utviklingsprosessen ble smidige metodologier brukt, med korte sprinter og hyppig tilbakemelding fra oppdragsgiver. En risikovurdering av løsningen ble utført ved hjelp av oppdragsgivers rammeverk, og både løsningen og risikovurderingen ble evaluert av eksperter hos oppdragsgiver. Brukertesting ble også utført.

Løsningen består av en nettverksoppstartstjener som bruker UEFI HTTP BOOT, en DNS-server for domenenavnoppslag, en DHCP-server for nettverkstilgang og oppstartsalternativer, en selvutviklet orkestreringstjeneste for å håndtere prosessen, og en Ansible Automation Platform-instans for klientkonfigurasjon etter OS installasjonen. Risikovurderingen indikerte at løsningen opprettholder et akseptabelt lavt risikonivå. Eksperter evaluerte og bekreftet at både løsningen og risikovurderingen oppfyller oppdragsgivers krav.

Brukertesting, ekspertvurdering og personlig erfaring dannet grunnlaget for fremtidig arbeid, som bør prioritere implementering av ytterligere sikkerhetstiltak, en risikovurdering tilpasset hvert miljø, og testing av tanking av flere klienter samtidig.

# PREFACE

Here is the bachelor thesis ”Streamlining the Provisioning of Physical Clients: A Secure Approach for an Environment with Low Risk-Acceptance”. It was written as the final work for our bachelor’s degree in the study program ”Bachelor in Computer Science”, at the Department of Computer Science, at NTNU in Gjøvik. We were looking for a project with a focus on information security and found one with the employer, who is not mentioned by name in this thesis.

Throughout this project, we were able to use much of our existing knowledge, especially regarding digital infrastructure, client configuration with code, and communication over private networks. However, we acquired even more knowledge, particularly when it comes to network booting, Dynamic Host Configuration Protocol options, and operating system configuration through Kickstart.

We would like to thank our supervisor at the employer, who from the beginning has been as if not more, invested in the project as us. You have provided us with good working conditions at your company, with access to all the equipment and knowledge we have needed. Thank you also to the other people at the employer who have contributed with user tests, expert evaluations, sparring, answers, and other help.

We would also like to thank our supervisor at NTNU, Kiran Raja. You made sure we kept our focus on the academic aspect and gave us strict, necessary deadlines. Thank you also to the IT service at NTNU, who gladly provided equipment and rackspace when we needed them.

We hope you enjoy your reading.

Vegard Færgestad
Mathias Westby Skoglund
Karishma Sharma
Gjøvik, May 17th 2023.

# Table of Contents

# Glossary

**API** Application Programming Interface: Software interface that allows two or more applications to talk to each other. 11, 13, 18, 19, 25, 26, 28, 44, 48, 55, 63, 68, 71, 74, 75

**Attack Surface** The collection of points or elements in a system that are exposed to potential threats or vulnerabilities.. 22, 24, 53, 58, 61, 66–71

**Attack Vector** Method or pathway used to breach the security of a system. 42, 66

**BIOS** Basic Input/Output System: Low-level software that initializes and manages the communication between a computer's hardware components and its operating system. It's responsible for performing the power-on self-test (POST) and bootstrapping the system during startup. 6, 13, 18, 41, 53, 54, 58, 59, 66

**Confidentiality** Principle of ensuring that data or information is accessible only to authorized individuals. 4, 15, 41, 70

**Countermeasure** Prevents threats from exploiting vulnerabilities. 76

**CURL** Command-line tool and library that allows users to transfer data using various protocols, such as HTTP, HTTPS, FTP, and more. 30

**DAC** Discretionary access control: Principle which restricts object access based on the identity of the subject, which refers to either the user or the group that the user belongs to. 10

**Declarative Programming** Defines the desired outcome of a program without specifying how to accomplish it. 10

**DHCP** Dynamic Host Configuration Protocol: A client/server protocol that assigns IP addresses and related configuration information to devices on a network. 11–13, 16, 21, 24, 25, 38, 41, 51, 52, 61, 64–66, 68–70, 74

**DNS** Domain Name System: Translates domain names into IP addresses. 11, 12, 16, 21, 24, 25, 38, 41, 51, 61, 64–66, 69, 70, 74, 83

**DNSSEC** Domain Name System Security Extensions: Suite of security protocols designed to protect the integrity and authenticity of DNS data. It adds digital signatures to DNS records, ensuring that the information provided by a DNS resolver has not been tampered with or altered during transit. 25

**firewalld** Open-source, dynamic firewall manager for Linux systems that provides a user-friendly interface for managing network security. It simplifies the configuration of firewall rules by using predefined zones and services, allowing users to easily define and control incoming and outgoing network traffic. 23, 25

**Go** Open source, high-level programming language designed by Google. 21, 25, 27, 29, 46, 48, 49, 61, 68, 70–72, 74, 75

**HTTP** Hypertext Transfer Protocol: Protocol for transferring information between devices on a network. 9, 15, 22, 24, 25, 27, 36, 38, 45, 47, 53, 59, 61, 63, 68, 69, 72, 75

**HTTPS** Hypertext Transfer Protocol Secure: Extension of the HTTP, using encryption for secure communication over a computer network. 6, 9, 15, 16, 18, 22, 23, 51–53, 57, 61, 72, 73, 83

**Integrity** Principle of protection of data and information from unauthorized modification. 7, 25, 57, 58, 60, 61, 73

**ISO-image** Optical Disc Image: Digital file that contains an exact copy of the contents of an optical disc, such as a CD, DVD, or Blu-ray. It is a widely used format for distributing software, operating systems, and other large files. 2, 7, 16, 18, 37, 38, 52, 59–61, 63, 64

**Linting** Process of analyzing source code to detect and report programming errors, stylistic issues, or non-compliant code patterns. 48, 49, 61

**LSM** Linux Security Modules: Framework in the Linux kernel that supports different security models. Provides a flexible and lightweight mechanism for implementing various access control schemes without favoring a specific implementation. 10

**MAC** Mandatory access control, restricts access to resources based on the sensitivity of the information and the user's authorization level. Security labels define the sensitivity of the resource and users can only access information within their security label's authority. 10

**Mocks** Process of creating fake or simulated versions of complex dependencies. 45

**OpenSSH** Open source connectivity tool for remote access to servers using the Secure Shell protocol. 10

**Public Key Infrastructure** Security framework that uses digital certificates, public and private keys, and certificate authorities (CAs) to enable secure communication, authentication, and data encryption over networks. 7, 60, 61

**PXE** Preboot Execution Environment: Industry standard created by Intel which provides pre-boot services within the devices firmware that enables devices to download network boot programs to client computers. 12, 13

**Red Hat** American company that delivers open source software solutions such as Ansible, Red Hat Enterprise Linux and OpenShift. 9–11, 13, 18, 20, 21, 23, 24, 36, 37, 57–60, 67–72

**REST** Representational State Transfer: Software framework that imposes conditions on how an API should work. Created as a guideline to manage communication over networks. 11, 13, 18, 25, 26, 28, 63, 68, 71

**SSH** Cryptographic network protocol used for secure communication between a client and a server over an unsecured network. 11, 36

**SSL** Secure Socket Layer: The predecessor to TLS for encrypting network traffic. 22, 62

**systemd** Modern, feature-rich init system and service manager for Linux operating systems. It is responsible for bootstrapping the user space, managing system processes, and controlling system services (daemons). 25, 30

**Threat** Event or action which has the potential to cause loss or harm. 41, 44, 76

**Threat Actor** Person or group that carries out or has the intention to carry out harmful actions against computer systems or data. 15

**TLS** Transport Layer Security: Cryptographic protocol designed to provide encryption over a computer network. 15, 18, 22, 24, 27, 52, 57, 61, 68, 72, 73

**UUID** Universally Unique IDentifier: Alphanumeric string which can be used to identify information. 26, 29, 30

**Vulnerability** Weakness that could be exploited to cause harm. 66

# Acronyms

**AAP** Ansible Automation Platform. 11, 18, 19, 26, 28, 33, 36, 38, 41, 45, 55, 58, 61, 63–65, 70, 71, 75

**BIA** Business Impact Analysis. 2, 41, 42, 54, 75

**CIS** Center for Internet Security. 23, 50, 53, 72

**CLI** Command Line Interface. 49

**FQDN** Fully Qualified Domain Name. 51

**IaC** Infrastructure as Code. 3, 10, 15, 47

**ISC** Internet Systems Consortium. 69

**JSON** Javascript Object Notation. 26

**NBP** Network Boot Program. 21, 23–25, 51, 54, 58, 61, 66, 69, 74

**OS** Operating System. 2, 6, 7, 9, 10, 12, 13, 15, 16, 19, 24, 30, 33, 36–38, 41, 52, 57–61, 63, 64, 83

**RHEL** Red Hat Enterprise Linux. 6, 7, 12, 13, 20, 22–24, 30, 36–38, 54, 58–60, 65, 67–69, 71, 73

**SARC** Security Architect. 33, 54, 75, 76

**SAST** Static Application Security Testing. 48, 63

**SSDLC** Secure System Development Life Cycle. 44, 82

**TCP** Transmission Control Protocol. 23, 25, 51

**TFTP** Trivial File Transfer Protocol. 12, 13

**UDP** User Datagram Protocol. 25

**UEFI** Unified Extensible Firmware Interface. 6, 9, 13, 15, 16, 18, 24, 47, 53, 55, 57, 59, 61, 63, 68, 72, 74

**VLAN** Virtual Local Area Network. 37

**YAML** YAML ain't markup language. 11, 12, 29

# List of Figures

# List of Tables

# List of Listings

# 1 Introduction

This section presents the background of this project, including a description of the problem and an outline of the project goals, scope, and constraints. Furthermore, it identifies the target audience and the organization of the group members. Lastly, the section offers an overview of the thesis structure.

## 1.1 Background

The employer for this thesis[1] frequently encounters the need to configure new laptops, hereafter referred to as clients, for various purposes such as onboarding new employees or upgrading existing clients. The current procedure for provisioning clients involves a substantial amount of manual work and time consumption. This involves locating the appropriate operating system, acquiring the correct configuration, transferring it to a physical medium, and subsequently configuring each client individually.

During this process, the employee must choose accurate options, input correct passwords, initiate configuration processes, and manually validate the installation. The provisioning process generally spans 4 to 5 hours per client. Given the non-standardized nature of this process, it is resource-intensive and prone to misconfigurations. In an environment characterized by strict security requirements and minimal risk acceptance, there is a pressing need for an improved alternative to the existing solution.

## 1.2 Problem Description

There must be developed a system capable of provisioning physical clients more effectively than the current approach. This system must be standardized and automated to minimize resource usage and misconfigurations while simultaneously provisioning multiple clients. Nevertheless, the system must maintain or surpass the existing solution's security and risk management standards.

## 1.3 Project Goals

The overarching goal of this project is to streamline the process of provisioning physical clients in an environment with low risk acceptance. To achieve the overarching goal, the group has established two subcategories of goals: result goals and effect goals. Result goals are specific goals that the group wants to accomplish through the solution. Effect goals, on the other hand, are intended to be achieved in the long term as an outcome of the solution.

### 1.3.1 Result Goals

- Provisioned clients are automatically booted, installed and configured.
- The clients are provisioned more consistently than doing the task manually.
- The provisioning system is developed and security assessed based on the framework provided by the employer.
- The provisioning system is able to scale when a large number of clients are needed.
- The provisioning process requires a minimum amount of human interaction.
- The provisioning process is less time-consuming than the current process.

[1] For an introduction to the employer, see Appendix A. The employer is not disclosed to the public.

### 1.3.2 Effect Goals

- Reduction of required human resources associated with provisioning clients.
- It is less time-consuming to maintain and manage changes to clients.
- The provisioning system is scalable and easy to maintain.
- The security of the environment and provisioning process is preserved or increased.

## 1.4 Project Scope and Constraints

The scope of this project is to design and develop a system for automatic booting, installation and configuration of clients. This includes the planning and implementation of a secure and useable solution, as well as facilitating its deployment. Additionally, performing a security assessment of the system is in scope.

Several aspects are outside the scope of this project. First, the specific configuration details of the client after the Operating System (OS) installation will not be addressed. Second, the creation of the required infrastructure for the system deployment, such as virtual machines and networks, will be provided by the employer. Third, the physical access controls are also beyond the scope of this project. The retrieval of the OS is handled by the employer and is therefore out of scope.

### 1.4.1 Time Constraint

The timeline for this bachelor project spans from January 9th to May 22nd, 2023, a period within which both the solution and the report are fully developed and submitted.

### 1.4.2 Confidentiality Constraint

Due to the employer, certain elements of the project must be kept confidential and are therefore not disclosed to the public when the thesis is published. Consequently, some sections, which normally would be a part of the thesis, can be found in Appendix A, B, D, F, and I. To gain a comprehensive understanding, it is important to consider them as parts of the thesis.

## 1.5 Overview of Confidential Appendices

The following appendices contain sensitive information and will not be disclosed to the public:

- Appendix A: Introduction of Employer
- Appendix B: Justifications of BIA
- Appendix C: Business Impact Model (BIA)
- Appendix D: Threat Assessment
- Appendix E Security Risk Assessment
- Appendix F: Deployment at the Employer
- Appendix G: Security Assessment Evaluation
- Appendix H: Secure System Development Life Cycle (SSDLC) Checklist

- Appendix I: Employer Pipeline
- Appendix J: User Testing
- Appendix K: Project Plan
- Appendix L: Project Contract
- Appendix M: Meeting Minutes

## 1.6 Target Audience

This bachelor thesis is intended for professionals, organizations, and students with an interest in software development, Infrastructure as Code (IaC), and cybersecurity. In addition, the project is relevant to those who want to incorporate similar functionality in their provisioning process. To better understand the technical aspects covered in the thesis, it is recommended to have basic knowledge of programming, computer networks, infrastructure, and basic security principles.

## 1.7 Organization

During the project planning phase, the organization of the project was determined, including group roles and responsibilities. The key information regarding these aspects will be provided here, while the complete plan can be found in Appendix K. In addition, the project involves external parties, each with their own distinct roles, which will be described in this section.

### 1.7.1 Responsibilities and Roles

The group members have been given roles and different responsibilities during the course of the project (see Figure 1). As a result, this improved time management, division of tasks, increased motivation and gave a sense of responsibility to the group members. This also helped improve the efficiency and overall success of the project.

- **Vegard Færgestad (Project leader):**

  The primary responsibility was to ensure that the project was progressing as planned and that all group members were actively engaged and fully understood their tasks. Additionally, the project leader served as the final decision-maker in the event of conflicts within the group. Furthermore, the project leader acted as a moderator during meetings, ensuring that discussions remain focused and productive. In the event that the project leader was unable to attend a meeting, this responsibility fell to the communication responsible.

- **Mathias Westby Skoglund (Technical Leader and Documentation responsible):**

  Responsible for providing technical direction and oversight for the project. This included the responsibility for the development of technical design and architecture, as well as being responsible for organizing code reviews to guarantee that the final product is of high quality and aligns with the project objectives.

  The group member was also responsible for the backup and overall structure of all documentation. This included ensuring that the documentation was complete, accurate, and consistent in terms of formatting and presentation.

- **Karishma Sharma (Communication responsible):**

Responsible for effective communication with external parties, and maintaining a clear record of all communication related to the project. Additionally, the communication responsible was responsible for distributing meeting minutes to external parties after each weekly status meeting, to ensure that all parties were informed of the project's progress and any necessary actions.

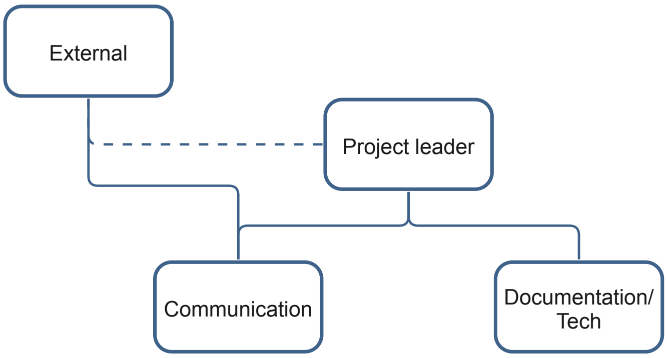


Figure 1: The project role structure of the group.

All group members were considered developers in this project and were expected to contribute accordingly to the research and implementation of the project.

**The Supervisor from the employer** for this project is not disclosed to the public. However, their role included providing the requirements and expectations for the project, as well as assistance with technical aspects.

**The Academic Supervisor** for this project is Kiran Raja, who works at the Department of Computer Science[2] at NTNU. His role involved offering assistance with academic report writing and providing overall guidance throughout the project.

## 1.8 Thesis Structure

The thesis consists of eight sections. The task specification, relevant background and technologies are presented in Section 2. The description of the architecture, implementation, and development process can be found in Section 3. Section 4 details the deployment of the solution, including considerations, deployment environment, client support and others. Moreover, a comprehensive security assessment can be found in Section 5. Section 6 describes the various measures the group has employed to ensure the quality of the project. Section 7 includes the discussion of the project, such as technical results, architecture, implementation, and the project process. It also discusses the user tests and security assessments conducted. Section 8 provides the conclusion, where it is discussed if the project's objectives have been achieved and a summary of further work is presented. Lastly, the relevant appendices are attached. Due to the confidentiality constraint, some appendices have been excluded from this document.

[2] https://www.ntnu.edu/idi, visited 06.05.2023

# 2 Task Specification and Related Work

This section presents the specifications for the solution, along with a section that aims to provide sufficient background knowledge by explaining key concepts and technologies relevant to the thesis. Furthermore, similar existing solutions will be introduced and evaluated, offering insights into the background and context of the project.

## 2.1 Task Specifications

The provisioning system must fulfil specific requirements set by the employer. These are related to both the system components themselves, and the provisioning of a client with the desired features. These requirements are divided into functional, non-functional, and operational specifications, collectively setting the basis for the solution.

### 2.1.1 Functional Specifications

The functional specifications describe what functions the provisioning system is supposed to have. They are split into two priority levels. The first level includes features that must be present to give the primary desired functionality of the system, while the second level includes specifications that are less important, but desired features.

**Priority 1**

- Secure network booting of physical clients.
  - Must utilize UEFI HTTPS boot.
  - Must support Dell clients.
- Automatic installation of Red Hat Enterprise Linux (RHEL) 9.
- Automatic execution of Ansible playbooks to configure the client.
- Automatic verification of the Operating System install.
- Non-critical errors should be logged, whilst critical errors should stop the provisioning process.

**Priority 2**

- The automated installation should be backwards compatible with version 8 of RHEL.
- The system should support reprovisioning clients.
  - The data from the previous provisioning should be securely removed.
- Automatic configuration of the BIOS.

### 2.1.2 Non-functional Specifications

The following section contains a list of non-functional specifications of the provisioning system. These are requirements that describe how the system should behave in terms of characteristics and attributes, not what functions the system should have.

**Deployment**

- The preferred deployment type is on one or multiple virtual machines (VMs).

- The system should be deployable on RHEL version 9.x.
- The deployed system should function without an internet connection.

**Security and Privacy**

- Code should be analysed by the employer's internal code pipeline.
- The system should be security assessed.
- The image of the Operating System (OS ISO-image) should be fetched from an approved secure location.
- Network traffic that can be encrypted, should be encrypted.
- A Public Key Infrastructure (PKI) should be used.
- Access to the system should be sufficiently secured.
- The integrity of the scripts used to configure clients should be ensured.
- The system should use secure default values.

**Usability and Maintainability**

- The configuration of the system should be defined in code using Ansible.
- The system should be scalable, preferably horizontally.
- The system should be documented sufficiently.
- The code should follow best practices and be easily maintainable.
- The system should make it easy to add new provisioning configurations.

**Reliability**

- The system should handle multiple provisionings at the same time.
- Every client should be provisioned consistently.
  - Detected discrepancies should be logged.

**Performance**

- The provisioning should take less time than the current manual duration.

### 2.1.3 Operational Specifications

Operational specifications refer to the requirements that should be followed when the system is running in production. These specifications should not be confused with deployment specifications that specify how the system is deployed.

- The system should create audit logs.
- The OS ISO-image file name and file hash should be logged upon provisioning.

## 2.2 Use Cases

The employer has two primary use cases that the solution aims to resolve. The use cases have the same actor; a provisioning coordinator who is responsible for managing the provisioning. The use cases can be found in Table 1 and 2.

| USE CASE | **Automatically provision many clients at the same time** |
|---|---|
| Goal | Provision clients without using unnecessary resources and time |
| Actor | Provisioning coordinator |
| Preconditions | Need for provisioning many clients at the same time, such as replacing outdated hardware, software updates or changes in policies |
| Success End Condition | Clients are provisioned consistently, rapidly and with minimal human interaction |
| Failed End Condition | Time consuming provisioning process with manual steps and unnecessary use of resources, or errors during provisioning |
| Description | 1. Prepare the clients<br>2. Start the automated provisioning<br>3. Monitor the provisioning<br>4. Disconnect the client when completed |

Table 1: Use case for provisioning many clients simultaneously.

| USE CASE | **Provisioning of client for new employees** |
|---|---|
| Goal | Provision clients without using unnecessary resources and time |
| Actor | Provisioning coordinator |
| Preconditions | Need for provisioning client(s) for new employees |
| Success End Condition | Clients are provisioned consistently, rapidly and with minimal human interaction |
| Failed End Condition | Time consuming provisioning process with manual steps and unnecessary use of resources, or errors during provisioning |
| Description | 1. Prepare the client(s)<br>2. Start the automated provisioning<br>3. Monitor the provisioning<br>4. Disconnect the client when completed |

Table 2: Use case for provisioning a client for a new employee.

## 2.3 User Stories

The automatic provisioning process involves one actor, namely the provisioning coordinator, who is responsible for managing the provisioning process. To better understand the system from an end-user perspective, the following user stories help identify the required functionalities:

- As a provisioning coordinator, I want to start the automatic provision on one or more clients, so that I can avoid simple manual tasks and increase productivity.
- As a provisioning coordinator, I want a clear and accurate display of the current status of the provisioning of the client, so that I can track its progress and identify potential issues that may arise.
- As a provisioning coordinator, I want to choose the desired provision configuration, so that the client is configured as wanted and expected.
- As a provisioning coordinator, I want to physically connect the client to the network, so that the client can communicate with the system.

- As a provisioning coordinator, I want to have access to a sufficient log of the provisioning process, so that I can monitor the success and failure rates and troubleshoot issues and errors.

## 2.4 Background and Technologies

This section provides a background for the project. It provides a description of some software applications, tools, and other relevant technologies. By doing so, it aims to offer a sufficient understanding needed for the thesis.

### 2.4.1 Kickstart File

Kickstart is an automated process to install a Red Hat based Operating System, where partial or complete requirements can be specified, and the desired configurations in a file [7]. This is a text file which uses keywords to specify elements, such as language and keyboard settings, users, and PRE- and POST-installation scripts. The system reads the information at boot time, and with complete requirements, the process is done without any further user inputs. The Kickstart installation can be run from different sources such as a DVD drive, a hard drive, or a Hypertext Transfer Protocol (HTTP)/Hypertext Transfer Secure (HTTPS) server [41]. If stored on a centralized server, the Kickstart file can be accessed by multiple individual computers during the installation process. This approach enables the use of only one Kickstart file to automate the installation of the OS on multiple machines, which is particularly useful for network and system administrators who need to install many systems with the same settings [71].

### 2.4.2 Network booting

Network booting uses a client-server approach to boot a computer using a computer network, instead of using bootable physical media, such as USB drives or DVDs [13]. One of the benefits of network booting is that, unlike other installation methods, no physical boot media is necessary. By deploying the installation source over a network, multiple systems can be installed from a single source without needing to connect and disconnect physical media [23]. A typical process of network booting is illustrated in Figure 2.

### 2.4.3 UEFI

The Unified Extensible Firmware Interface (UEFI) specification is a framework for booting a computer's OS and running pre-boot applications. By defining the interface between the Operating System and platform firmware, UEFI introduces a set of data tables containing platform-specific information, as well as boot and runtime service calls that can be utilized by the OS and its bootloader. Together, these elements create a standardized environment that enables an OS to boot successfully and execute pre-boot applications.

EFI files, or Extensible Firmware Interface files, are a key part of the UEFI system. They are small binary executables that contain the necessary instructions for initializing hardware components and starting the boot process [70].

In essence, UEFI provides a standardized approach for booting up a system and executing the necessary software for it to function [70]. The UEFI firmware provides various boot modes, one of which can be HTTP boot. This boot mode utilizes HTTP to enable network-based system booting [66].

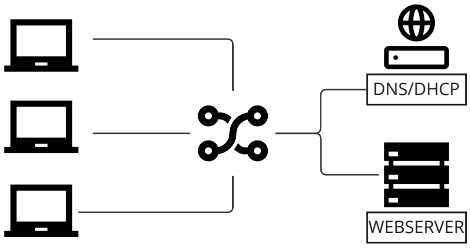


Figure 2: Example of network booting

### 2.4.4 SELinux

SELinux is a security architecture in the Linux kernel that enables administrators to more fine-grained access control [75]. Developed by the US National Security Agency (NSA) using Linux Security Modules (LSM), SELinux defines access controls for the system, including applications, processes, and files. It is done through security policies, which specify the rules for access allowed. When an application requests access to an object, SELinux checks the access vector cache for cached permissions. If SELinux cannot decide based on cached permissions, it checks the security context of the app or process and the file. Permission is then granted or denied based on the SELinux policy database. SELinux operates as a labelling system, where every file, process, and port on a system is assigned an SELinux label. Linux and UNIX systems have typically used Discretionary Access Control DAC, but SELinux was created as a Mandatory Access Control MAC system specifically for Linux.

### 2.4.5 Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a concept that aims to manage and automate the provisioning of infrastructure through code rather than manual processes [73]. With IaC, developers create code files containing their infrastructure specifications, making it easier to modify and distribute configurations. This approach also ensures idempotency and helps avoid undocumented configuration changes. Automating infrastructure provisioning enables the system administrator to avoid manually provisioning and managing servers, Operating Systems, storage, and other infrastructure components each time a new application or service is to be deployed. Codifying infrastructure provides a template for provisioning and enables developers to divide infrastructure into modular components that can be combined in various ways through automation. By improving consistency, reducing errors and manual configuration tasks, IaC is able to strengthen IT infrastructure management. Configuration management tools in IaC can employ either a push or pull approach to apply configurations to remote systems. In the push model, the configuration is ”pushed” from a central controller to the target systems. In contrast, in the pull model, the target systems ”pull” the configuration from a remote system [64].

### 2.4.6 Ansible

Ansible is a Python-based command-line configuration management tool designed for IT automation. It is an open-source tool, maintained by Red Hat, that can perform various tasks, such as configure systems, and orchestrate workflows to support application deployment and system updates [29]. The main advantages of Ansible are its simplicity and ease of use. It uses a human-readable language and follows a declarative programming approach to IaC. Ansible emphasizes security and reliability, with minimal moving parts, and relies on OpenSSH for connecting to the host machines [3]. Moreover, Ansible integrates with different tools for infrastructure, networks, containers, cloud, DevOps, and security. With its focus on simplicity and security, Ansible is a popular choice for IT automation in various industries. Ansible uses the push technique (see Section 2.4.5). This is accomplished by executing Ansible playbooks on the target hosts. Playbooks are written in YAML[3] format and contain a list of tasks to be executed on a group of hosts defined in the Ansible inventory. These tasks can be grouped and defined as an Ansible role, which can be used in playbooks. Ansible playbooks can also include conditions and variables to enable more complex and flexible configurations [72].

### 2.4.7 Ansible Automation Platform

Ansible Automation Platform (AAP) created by Red Hat is designed to facilitate IT automation. It is an integrated enterprise solution for operationalizing Ansible [63]. With this platform, users across an organization can easily create, share, and manage automation. Role-based access control is an essential feature of AAP, allowing control over resources like the securely stored credentials for SSH and other services. The platform logs all jobs, integrates with authentication sources, and offers a controller with a web console and REST API, execution environments, analytics, and other advanced features [4].

### 2.4.8 dnsmasq

dnsmasq is an open-source and lightweight DNS and DHCP implementation [40], well-suited for resource constrained routers and firewalls. It provides an integration between the DHCP and DNS services, enabling machines with DHCP-assigned addresses to be listed in the DNS [14]. dnsmasq supports both static and dynamic DHCP leases, meaning IP addresses can be manually configured by system administrators or automatically leased from the DHCP server [16].

### 2.4.9 NGINX

NGINX is an open-source software application primarily designed to function as a web server, acknowledged for its ability to deliver high levels of performance, scalability, and reliability. Originally developed by Igor Sysoev in 2002, NGINX has become one of the most widely used web servers in the world, powering millions of websites and web applications[53]. It can be used as a standalone web server or as a reverse proxy and load balancer. In addition to its core web server functionality, NGINX also includes a range of additional features and modules, such as caching, compression, and security features, which can be used to further enhance the performance and security of web applications [54].

## 2.5 Related Work

The aim of this section is to provide relevant background by evaluating and exploring existing similar provisioning solutions. This section will only consider solutions that are publicly available. After the state-of-the-art review, the group did not come across any research, applications, or solutions that satisfy the functional specifications that this system is required to have. However, the group acknowledges the possibility of other similar solutions that may not be publicly available, and hence, not verifiable. The following solutions have been evaluated: Metal as a Service, Microsoft Intune and Red Hat Satellite.

[3] https://yaml.org/

### 2.5.1 Existing Solutions

Metal as a Service (MaaS) is an open-source server provisioning system maintained by Canonical[4]. It aims to be a self-service remote installation tool for Ubuntu, CentOS, Windows and RHEL. However, it does not exclude other Operating Systems. Its primary use case is to automatically provision bare-metal servers, to make them act as a bare-metal cloud. This is primarily accomplished by using Trivial File Transfer Protocol (TFTP) and Preboot Execution EnvironmentPXE booting to automatically install the desired OS. Another feature of MaaS is automatic network discovery to detect computers on the network for easier provisioning. In addition, it keeps a detailed inventory of the computers it manages. The product can also act as a DNS and DHCP server for the rest of the network. This makes the provisioning process easier since it automatically can provide the correct DHCP options for network booting. Some benefits of using MaaS are the ease of use, speed of deployment and automatic hardware testing [46]. MaaS only supports TFTP booting, which might be deemed insecure, depending on the accepted risk levels of an organization [69].

One benefit of using MaaS is its integration with Juju. Juju is a high-level tool that helps manage applications on servers, virtual machines and Kubernetes. The tool aims to effortlessly manage applications across cloud, bare-metal and hybrid cloud infrastructure. Canonical claims that Juju is an application management tool, which they feel is a better infrastructure management concept and abstraction than configuration management tools. Juju uses Charmed Operators ("Charms"), which is a small application to package common maintenance functions together with the application [38]. It also provides an SDK that can be used to package the application into the common ".charm" file [39]. It aims to avoid making the system administrator easily deploy applications without mainly relying on YAML, which other similar tools rely on [74]. Using Juju in combination with MaaS, it is possible to have a fully or partially automated provisioning process [46].

Microsoft Intune is a service that enables organizations to manage their mobile devices, applications, and data. It consists of several cloud-based Microsoft services that streamline the deployment and management of devices across an organization. The tight integration with Microsoft services makes Intune an ideal way of deploying Microsoft Windows-based machines [48]. Microsoft Autopilot is one of these services, which is a device provisioning and deployment service for Windows 10 and 11. During the initial deployment of new Windows devices, Autopilot uses the preinstalled Original Equipment Manufacturer (OEM) image and drivers of the device. This functionality makes the users able to get their devices in a business-ready state with a fast and simple process. Autopilot provides different services based on the scenario of deployment, including user-driven mode, self-deployment mode and pre-provisioning mode [76].

In the user-driven mode, the user chooses language, locale, and keyboard, and specifies the credentials of their organization account. After connecting to a wireless or wired network with an internet connection, the rest of the process is automated. The device is configured based on the specifications defined by the organization [80]. In the self-deployment mode, there is little to no interaction from the user. Devices with Ethernet connection required no user interaction. If the user connected is to Wi-Fi; language, locale, and keyboard settings need to be chosen. The rest of the deployment is done automatically after making a successful network connection [79]. The pre-provisioning mode is a process divided into different stages. The time-consuming parts of the provisioning process are done by IT, partners, and/or OEM. It is only necessary for the user to configure a few necessary settings and policies [77]. What is common for all modes is that they enable support for Active Directory[5] and Microsoft Intune. Furthermore, Autopilot also has a reset functionality which takes the device back to a business-ready state, by removing all personal files and settings. However, it keeps the Wi-Fi connection

[4] https://canonical.com, visited 18.05.2023

[5] https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview, visited 18.05.2023.

details and other configurations, such as provisioning packages previously applied and Active Directory membership information [78]. The integration between Autopilot and Intune allows organizations to provision and deploy devices with minimal effort, while remaining compliant with corporate policies and security requirements [30].

Red Hat Satellite is an infrastructure management tool that is specifically designed to optimize RHEL environments and other Red Hat infrastructure, ensuring that they run efficiently and remain secure and compliant with various industry standards [8]. The tool offers a range of features, including external content sources like git and software package repositories, a Satellite Server for managing the life cycle of hosts, and Capsule Servers that mirror content from the Satellite Server to establish content sources in different geographical locations. The latter allows host systems to pull content from servers in their respective locations, rather than relying solely on the centralized Satellite Server. Additionally, Red Hat Satellite features an inspecting and reporting tool that can identify and report environmental data, such as Operating Systems and configuration data[35]. Red Hat Satellite consists of several open-source projects, including Foreman, Hammer, Katello and a REST API service [1].

Moreover, Red Hat Satellite provides several options for provisioning bare-metal instances, including BIOS and UEFI based PXE booting. These options define the DHCP filename option for use during the provisioning process and allow for downloading a file over TFTP to a client computer [2]. It is important to note that the provisioning capabilities of Red Hat Satellite are not limited to bare-metal instances, as the tool can also provision to virtual, private, and cloud environments.

### 2.5.2 Limitations of Existing Solutions

The evaluated solutions have limitations which do not satisfy the functional specifications. However, MaaS is close to the desired feature set when combined with Juju. The existing solutions were partially relevant; however, the services were not viable options due to the following reasons:

- Red Hat Satellite and MaaS use PXE and TFTP during booting
- Microsoft Intune relies heavily on cloud-based services for Windows (with no apparent support for Linux)
- Neither of the solutions provided a native and automatic way of running Ansible playbooks, which is a requirement given by the employer

These services also included additional features beyond the scope of this project. However, they contributed to the understanding of the potential opportunities and tools available for automated provisioning. The evaluation process also gave valuable insights into the provisioning market and the demand for such solutions.

# 3 Method and Process

The developed solution automatically provisions clients using a combination of components which collectively ensure each client is installed with an Operating System (OS), is configured, and that the entire process is logged. The solution is developed and deployed using Infrastructure as Code (IaC), featuring a modular design that enables adaptation to various environments. This section presents the main components of the project's execution, including a detailed description of the architecture and implementation of the solution. It also presents various considerations and decisions made during the process. Furthermore, the section explains how the development model was applied and provides a summary of the development process within the project.

## 3.1 Architecture

The following sections aim to provide an overview of the architecture by describing the various components, and how they interact with each other. Moreover, it will further detail certain elements of some components.

### 3.1.1 Components

To realise the task specifications, the system architecture has been divided into three primary components. A network booting component, a component dedicated to client configuration using Ansible playbooks, and an orchestration component to integrate the various stages of the provisioning process (see Figure 3).

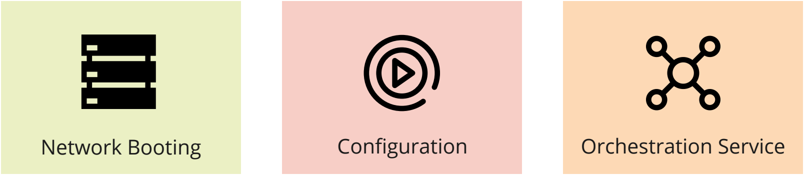


Figure 3: Overview of the components in the solution.

**Network Booting**
There are multiple methods that a client can use to perform network booting. For this solution, the requirement for the method was already decided by the employer. To minimize risk, UEFI HTTP booting was chosen as the ideal boot method. This UEFI feature lets a computer boot using files served by a regular web server. HTTP is unencrypted, but the use of HTTPS and TLS certificates solves this confidentiality challenge. With a TLS certificate, the network traffic between the server and the client is encrypted and cannot easily be intercepted and viewed by a threat actor.

To facilitate network booting, the components consist of a web server serving necessary boot files, a DNS server providing domain resolution, and a DHCP server providing network and boot location (see Figure 4).

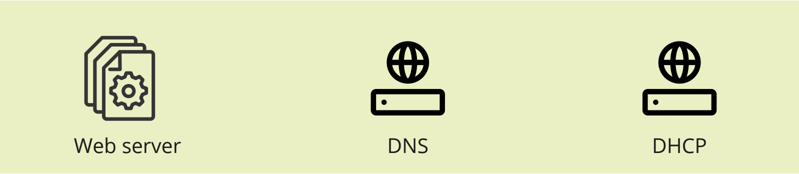


Figure 4: Overview of the parts in the network booting component.

The UEFI HTTP boot method requires the necessary files to be served from a reachable location. Consequently, the solution consists of a web server being responsible for serving these files. This includes an OS ISO-image, in addition to the files this specific solution requires; a Kickstart file, and a configuration file.

To make the system support multiple OS ISO-images and configurations, the user is shown a boot options menu when booting to the server. This menu is also retrieved from the web server (see Figure 5). This part of the system is interchangeable and must not necessarily be present for the system to work as intended. From this menu, a set of configuration options will be displayed. The selected option will then take care of which OS ISO-image to boot into, and which Kickstart file to use.

To ease the addition of a new configuration for the system administrator, a common file system structure for the web server has been defined as shown in Figure 5. It consists of a set of folders per boot option. Inside each of the folders, the OS ISO-image, the Kickstart file and the configuration file should be present.

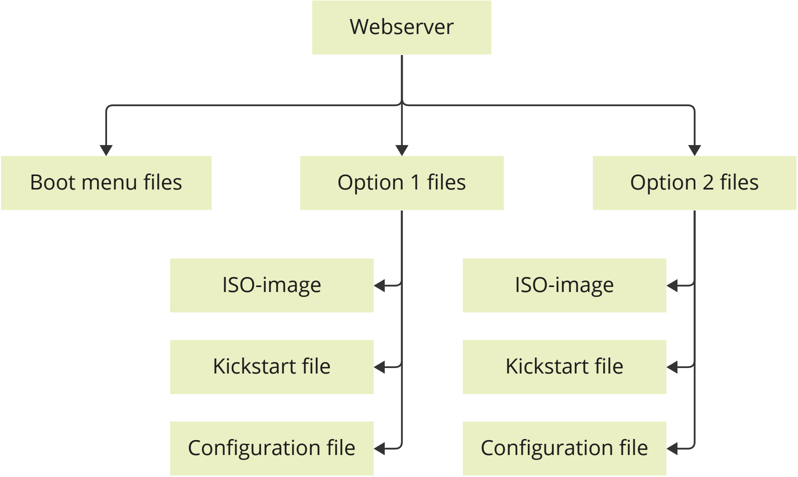


Figure 5: The folder structure on the web server.

To increase the usability of the HTTPS boot server, a DNS and DHCP server is a part of the network booting component. The DNS server enables reaching the web server through a human-readable name, instead of the IP address. The DHCP server has two main responsibilities. Firstly, it provides the clients with a network configuration, removing the need to set these manually. Secondly, it provides the desired boot file location to the client. This lets the user connect the client to the network and select network booting in the UEFI, for then to be automatically presented with the configuration options screen.

### The Orchestration Service

The orchestration service aims to integrate the various services and components together and provide the user with a status of the installation process. One of the key functionalities of this service is to ensure that the different processes run in the expected order, by keeping information on the stages. If one of the stages fails, it should be captured by the orchestration services and an error message explains the cause of failure to the user. The service is also responsible for creating and keeping logs of all these events (see Figure 6).

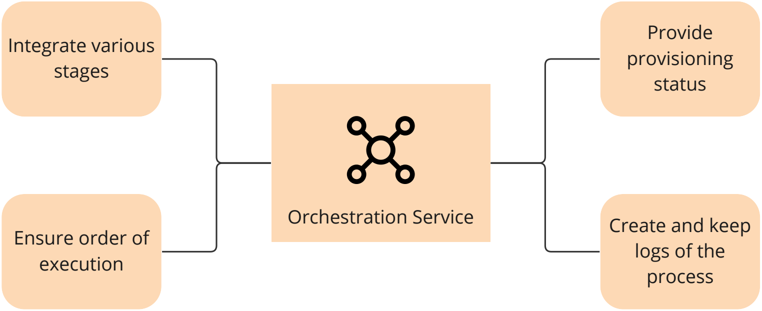

Integrate various stages
Ensure order of execution
Orchestration Service
Provide provisioning status
Create and keep logs of the process

Figure 6: The responsibilities of the orchestration service.

### Configuration

After the initial installation process, the client is configured automatically using Ansible. The provisioning system does not have this feature integrated; however, it is being facilitated via an integration to an existing configuration service that is able to run a set of Ansible playbooks (see Figure 7). This will ease the expanding or changing of the playbooks, compared to if they were bundled with the system. This also promotes reusability of the playbooks, since they do not have a tight coupling to the provisioning system.

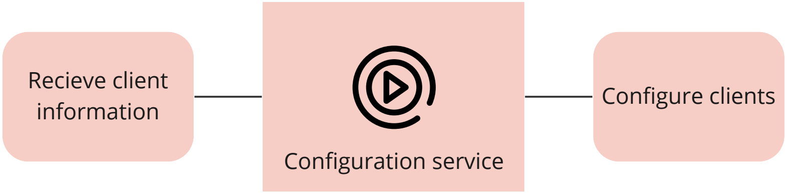


Figure 7: The responsibilities of the configuration service.

The components communicate with each other at multiple stages of the process. An overview of this communication can be seen in Figure 8.

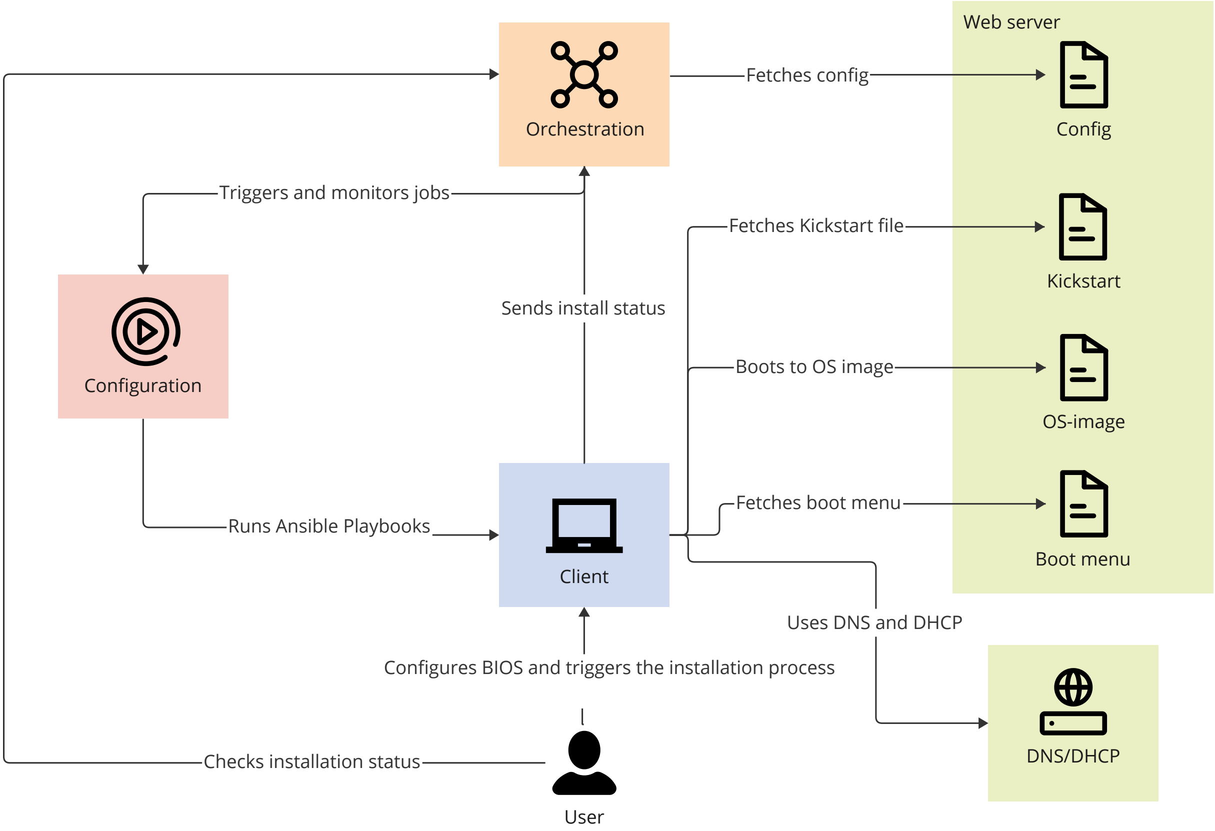


Figure 8: An overview of the high-level architecture.

Even though this solution aims to automate most stages, some manual steps still remain

At every new configuration:

- The ISO-image must be moved to the file server.
- The Kickstart file must be moved to the file server.
- The configuration file must be moved to the file server.

At every provision:

- The user must configure the UEFI BIOS of the client to enable needed features.
- The TLS certificate of the HTTPS must be added to the client to ensure trust in the TLS certificate on the HTTPS server.
- The user must choose the desired configuration option in the boot menu.

### 3.1.2 Ansible Automation Platform (AAP)

The chosen implementation of configuration service is Ansible Automation Platform (AAP). The clients are configured using Ansible playbooks as specified in Section 2.1. Since Ansible uses the ”Push” model for applying configuration (see Section 2.4.5), a centralised service is needed to easier execute the Ansible playbooks on the clients. The decision was made to utilize Ansible Automation Platform (AAP) [63], since it is created and maintained by Red Hat. AAP exposes a REST API which offers the service's functionality. This API is used by the orchestration service to create new hosts and inventories, in addition to executing existing playbooks on the clients. AAP is an important component of the architecture, but the deployment and configuration are not provided by the group. The deployment requirements can be found in Section 4.1.

### 3.1.3 The Orchestration Service

The solution had multiple specifications that created the need for an orchestration component to integrate the various stages that the client goes through during the provisioning process. This resulted in an orchestration service with several responsibilities.

Firstly, the service maintains an overview of the clients who are currently in the process of or have completed the provisioning. As the provisioning advances, it tracks the different states that the client transitions through (see Figure 9). This includes an error state, which enables the reporting of issues encountered during the provisioning process.

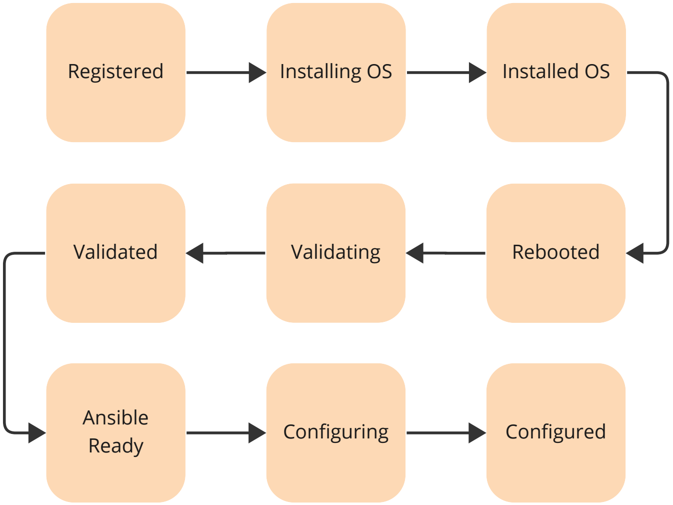


Figure 9: The different client states in order during successful provisioning.

Secondly, the orchestration service is responsible for retrieving the client configuration files provided by the web server. The configuration enables the fine-tuning of some aspects of the provisioning process by defining the playbooks to be executed on the client, and how to handle a failure. This ensures a more tailored provisioning process for each client.

Thirdly, the service handles the execution of the next stage when required. This occurs when the client is finished installing the OS, has rebooted, and reports that it is ready to start the configuration. The orchestration service starts the configuration by communicating with AAP. The service monitors what playbooks have succeeded and failed and finally sets the client in a configured or error state. The communication between the orchestration service and other components is visualised in Figure 10.

Lastly, the service is responsible for generating and maintaining logs throughout the provisioning process, logging both successful and error events. This documentation ensures effective troubleshooting, promotes transparency and enables analysis of the system's performance. This contributes to the improvement and optimization of the provisioning process.

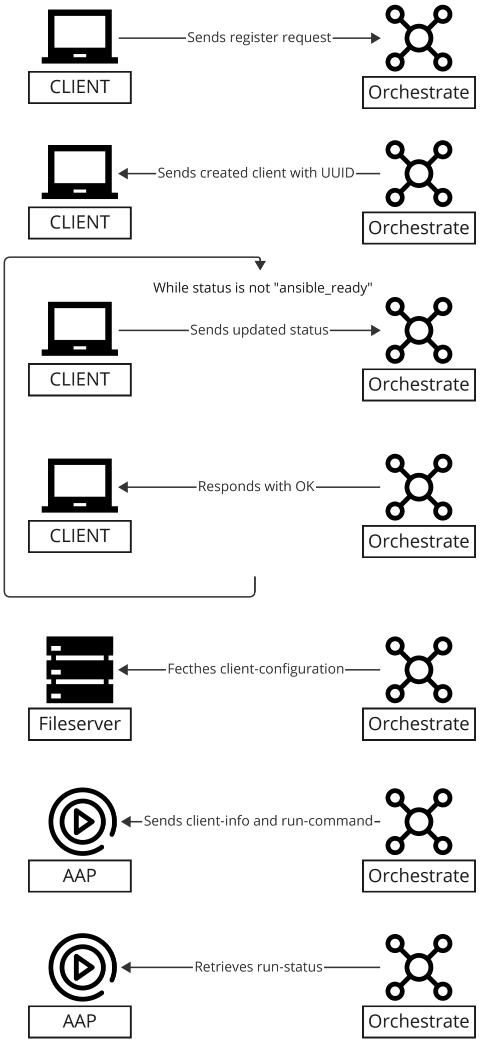


Figure 10: The communication between the orchestration service and other components throughout one provisioning.

The orchestration service needs a dedicated configuration file to function optimally. This allows for fine-tuning of how the service operates. This is explained in detail in Section 3.2.5.

## 3.2 Implementation

This section describes how the architecture has been fulfilled. It describes the set of tools needed to solve the different problems (see Table 3 for an overview), and how they are implemented. When deciding which tools to use, the following considerations were made:

- The tool should be open and freely available.
- The tool should be supported by the vendors the employer uses.
  - The tool should be maintained/supported by Red Hat[6].
  - The tool should be available in the Red Hat standard library.

[6] The majority of packages available in the RHEL standard repository will be maintained for the life cycle of the release by Red Hat[26].

- The group's familiarity with the tool.
- The tool should not introduce major risks to the environment.
- The tool should introduce as few dependencies as possible.
- The tool should have a minimal footprint.

| Name | Use case | Reasoning |
|---|---|---|
| NGINX | Web server | Wide spread use<br>Small footprint<br>Supported by Red Hat[24] |
| dnsmasq | DHCP/DNS | Wide spread use<br>Has the desired features<br>Only necessary with one package to serve both functions<br>Supported by Red Hat[24] |
| Ansible Automation Platform | Automated and centralized system for executing Ansible playbooks | Maintained by Red Hat[63]<br>Automatically handles secrets<br>Built for enterprise deployments of Ansible |
| Go | Programming language used for creating the orchestrating service | Modern language with a type system and garbage collection<br>Fast execution and compilation times<br>Code can be compiled into a static binary<br>Supported by Red Hat[24] |
| GRUB | Boot loader and Network Boot Program (NBP) (default boot location on the boot network) | Wide spread use<br>Simple<br>Supported by Red Hat[24] |

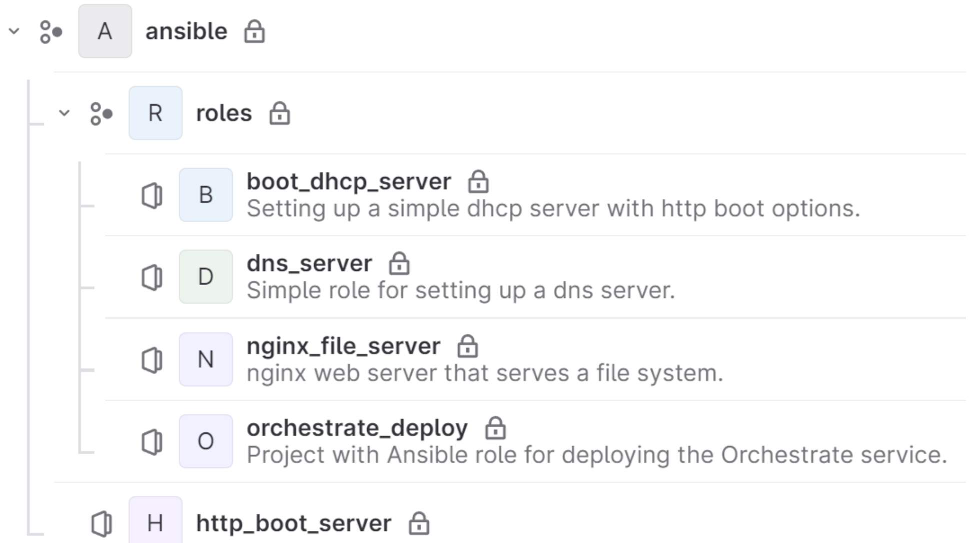
Table 3: The tools used in this project with reasoning.

### 3.2.1 Ansible Deployment Scripts

Every component of the provisioning system will be deployed and configured using Ansible as specified in Section 2.1. This will ensure consistency and act as documentation for the deployment of the system, as well as making it easier to perform changes for the system administrator. Additionally, Ansible supports using variables, enabling fine-tuning of the components.

The Ansible code is split into various roles of smaller standalone pieces. For instance, NGINX has a role for the base configuration of the service. For an overview of all roles and playbooks used to deploy the provisioning system, see Figure 11. In the roles, only the relevant and tuneable aspects are set using variables to assure flexibility. The actual instantiation and configuration of the roles are done in the `http_boot_server` repository, containing the primary Ansible playbook. This playbook configures all aspects of the provisioning system. This makes the installation and maintenance of the deployment easier. Inside the `http_boot_server` playbook the different functions are divided into various host groups, so they can be deployed on the desired hosts. In addition to being the playbook where all the roles are deployed and configured, it also deploys the GRUB[7] NBP and the custom boot menu.

[7] https://www.gnu.org/software/grub/, visited 17.05.2023.



ansible

roles

boot_dhcp_server
Setting up a simple dhcp server with http boot options.

dns_server
Simple role for setting up a dns server.

nginx_file_server
nginx web server that serves a file system.

orchestrate_deploy
Project with Ansible role for deploying the Orchestrate service.

http_boot_server

Figure 11: Overview of the different Ansible repositories in GitLab.

### 3.2.2 HTTP Web Server

When deciding what web server to use, the main desired features were: security, lightweightness and ease of use. These key traits are desired factors to minimize the attack surface and make sure the server is easily expandable. Making the deployment low maintenance was also an important factor. The web server NGINX was chosen to serve as the boot server. In addition to NGINX being part of the standard RHEL 9 repository [24], it is flexible and relatively easy to configure. The web server is configured and deployed by Ansible to act like a HTTP file server. It is set up to serve a single folder, where the boot configurations are present, as shown in Figure 5.

The NGINX HTTP file server is implemented in an Ansible role in its own git repository, making the module reusable for other purposes (see Figure 11). The role is split into three components; one to configure NGINX, one to configure the firewall on the host, and one to ensure the proper permissions on the web server folders. The implementation of NGINX has been done with security in mind. Following, some important aspects of the configuration are covered. For an example of a configuration, see Appendix N.

To decide which folder NGINX serves, the `root` option is specified. This, in combination with the `autoindex on;` option, makes the web server display all files and folders inside the specified root folder. This is necessary to make NGINX automatically act as a HTTP file server, including listing directories and their contents [55], further discussed in 7.4.1. Since the file server location is only to be used as a read-only file server, only the HTTP methods GET and HEAD are allowed. In addition, version 2 of HTTP has been enabled on this location for the speed and security improvements [56].the

To make sure that the traffic is encrypted, TLS has been enabled for the file server location. The system administrator must provide their own TLS certificate since it is not automatically generated by the role. The role supports all the SSL protocols that NGINX uses, through an Ansible variable. However, the default value of `TLSv1.2 TLSv1.3;` should be used for improved security. Regarding the `ssl_ciphers` parameter, it has been configured to a narrower range of cipher suites, compared to the NGINX default, yet still compatible with the Dell clients. This adjustment is due to the fact that various TLS versions continue to support a broad array of insecure ciphers [52][44]. In addition, an automatic HTTP to HTTPS redirect is enabled by default. This should make it impossible for a user to access the web server unencrypted.

In addition to these security measures, some headers are dictated by the NGINX server. The `Strict-Transport-Security` (HSTS) header makes sure that HTTPS is always used by the client, and it helps avoid HTTPS click-through prompts in the browser [59]. Moreover, the `X-Frame-Options` is applied with the `SAMEORGIN` option to tell the browser not to render the page if it is in a frame, iframe, embed or object HTML tag [51]. To help avoid common attacks like cross-site scripting and data injection, the `Content-Security-Policy` header is set; when set to `default-src 'self'`, only content from the main site is able to be loaded and no content from any other domains [50]. These headers are defined as best practice by CIS[8].

To ensure that the server is able to listen on the desired ports, a firewalld opening is required on the host. This is done through the `ansible.posix.firewalld` module. The `nginx_file_server` role sets a rich IPv4 rule[9] to accept TCP connection on the ports specified by the Ansible variables. To add extra security, there is provided an option to set which subnets the ports accept. This will ensure that no connections from other subnets than the one specified is allowed.

Which root folder is served by NGINX is specified by an Ansible variable (set to /var/www by default). If the folder does not exist, the role will create it. Even if the folder exists already, the file owner will be the nginx user and the nginx group. The folder permission is set to read and execute for the nginx user and group, but other users have no access. To make NGINX able to read the files, the correct SELinux context must be set. For the case of NGINX, the context should be `httpd_sys_content_t` [17]. The proper SELinux context must also be set for the network port to allow the traffic to flow as intended[10][22]. There is also a need to ensure that the NGINX process is set to the correct context. This is done automatically when installing the NGINX package through the RHEL 9 standard library [17].

In addition to the regular configuration of NGINX, some additional steps have been taken to harden the deployment. To ensure the quality of the hardening, the Center for Internet Security (CIS) NGINX benchmark was used. See Section 6.6.2 for more information regarding how it ensures quality and security, and Section 7.4.1 for the discussion regarding which recommendations have been applied.

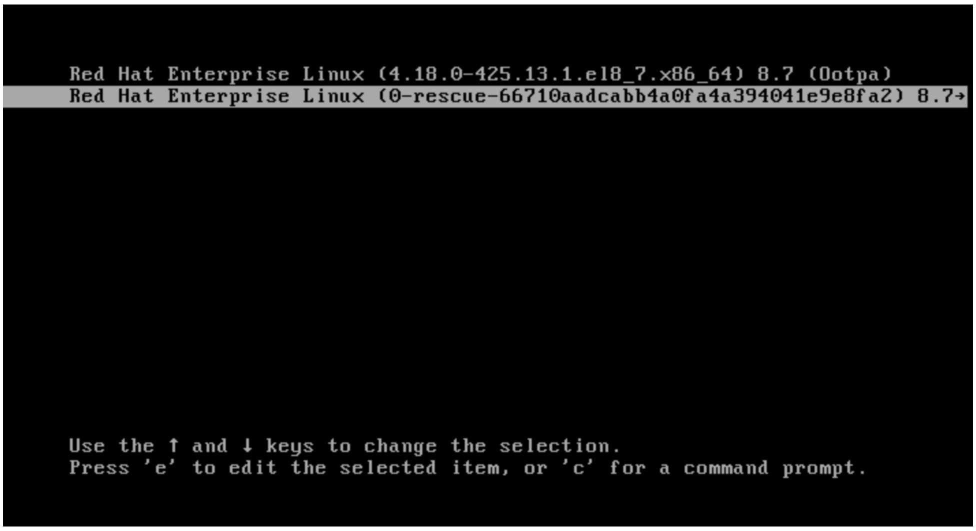
### 3.2.3 GRUB Network Boot Program (NBP)

The GRUB Network Boot Program (NBP) is deployed on the web server. This service provides the user with a configuration menu (as seen in Figure 12). GRUB was chosen because it is the default bootloader on most Linux servers and clients, and is therefore already present[5]. If the NBP were generated on a RHEL server, the configuration should be supported by Red Hat[24]. The deployment of the GRUB environment and configuration of the boot menu is automated with Ansible.

[8] Version 2.0.0 of the NGINX CIS benchmarkhttps://www.cisecurity.org/benchmark/nginx visited 13.05.2023.

[9] A type of rule in firewalld: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/configuring_complex_firewall_rules_with_the_rich-language_syntax, visited 20.05.2023.

[10] By default, the ports 80, 443, 488, 8008, 8009 and 8443 have the correct SELinux context set[22].



```
     Red Hat Enterprise Linux (4.18.0-425.13.1.el8_7.x86_64) 8.7 (Ootpa)
     Red Hat Enterprise Linux (0-rescue-66710aadcabb4a0fa4a394041e9e8fa2) 8.7→







     Use the ↑ and ↓ keys to change the selection.
     Press 'e' to edit the selected item, or 'c' for a command prompt.
```

Figure 12: An example grub menu.

The command `grub2-mknetdir --net-directory=. --modules=http` was used to create the GRUB NBP on a RHEL 9 UEFI system. The command creates a folder containing a boot environment which contains menu functionality. This allows the provisioning system to display a custom list of boot options. This list is generated by Ansible and only supports booting to EFI (UEFI) files. This is not an issue since systems using UEFI HTTP boot also supports installing Operating Systems in UEFI mode. Per the menu option the system administrator has to provide the following:

- Boot server hostname (including the schema)
- Configuration prefix (subfolder where the boot files are present)
- Path to the kernel and initrd[11] file (from the subfolder scope)
- Other kernel arguments that the system administrator desires

By default, the Ansible script adds some RHEL specific options. The option `inst.repo` and `inst.stage2` specifies where the kernel can get the other installation files needed during the installation. The `inst.ks` option specifies the Kickstart file location. This defaults to a `kickstart.cfg` file in the subfolder on the web server. Finally, the `inst.noverifyssl` options tell the kernel not to verify the TLS certificates.

### 3.2.4 DNS/DHCP

The package used for DNS and DHCP is dnsmasq, as it serves DHCP and DNS in a single package. This lowers the attack surface since fewer packages are involved. It is part of the RHEL standard library and is therefore supported by Red Hat [24]. The DNS server's primary job is to serve local DNS resolutions for the provisioning system and should therefore send external requests to an upstream DNS server. The DHCP server is configured for basic DHCP operations, such as setting the client's IP address, DNS server, and gateway. In addition, the necessary boot options are set for a UEFI HTTP boot client to automatically get the boot server and the boot file location. This ensures that a client automatically boots into the GRUB NBP.

Even though the DNS and the DHCP server uses the same package, dnsmasq, they are separated into two Ansible roles. This makes it easy to deploy only the components

[11] The initial ram disk that is available to the kernel before the actual file system is loaded [31].

```
# Http boot config
dhcp-vendorclass=httpclients,HTTPClient
dhcp-userclass=set:httpclients,HTTPClient
dhcp-option=60,HTTPClient
dhcp-option=tag:httpclients,option:bootfile-name,https://boot.
    group2.com/boot/grub/x86_64-efi/core.efi
```

Listing 1: Network booting specific DHCP configuration.

that are needed in a given deployment environment. However, dnsmasq has some global options which can only be set in one of the roles at the time. the instances of the roles. Both the Ansible roles install the dnsmasq package and configure the correct permissions. They also make sure that the systemd service is enabled and running. Since dnsmasq acts as a listening service, ports have to be opened in the firewalld configuration (the ports are detailed in Table 4).

| Port | Protocol | Service |
|---|---|---|
| 53 | TCP | DNS |
| 53 | UDP | DNS |
| 67 | UDP | DHCP |

Table 4: The default ports required for DNS and DHCP.

For the configuration of DNS, the system administrator has the ability to define an upstream DNS server, which enables the possibility to make the server reply to non-local DNS queries. An upstream DNS can for example be Google DNS or another internal DNS server. The dnsmasq configuration is set to always log all queries to the administrator-specified log location. Beyond that, DNSSEC is an optional parameter to ensure better integrity of the external DNS queries. When enabled, it uses the default dnsmasq trust anchors to verify that the DNS data is provided by a trusted source [32]. For extra security, the `dnssec-check-unsigned` option is enabled, so that dnsmasq checks that the unsigned upstream DNS replies are legitimate [58]. The integrity verification through DNSSEC is only applicable when using an upstream DNS server for external records. In addition, the Ansible role lets the administrator specify custom DNS entries. This makes it easy for the administrator to add local DNS record(s) for the boot server(s).

In contrast to the DNS server role, the DHCP server role is more specialized towards the provisioning system. As with the DNS server, all DHCP requests are logged to the specified log location. This makes it easier to investigate errors and to discover misuse of the system. The system administrator must specify the default gateway for the network and the desired DHCP range if the DHCP server is not set to be a DHCP proxy. Optionally, one or more DNS servers can be specified. The administrator also has the ability to set DHCP reservations based on MAC address if desired.

To automatically provide a HTTP boot server to a connected client, the DHCP options 60 and 67 are set. The DHCP option 60 specifies the vendor class identifier, which must be set to `HTTPClient` for most vendors. This ensures that it is possible for the client to HTTP boot using the provided boot file located in DHCP option 67. Option 67 makes it possible to specify a file name through DHCP [33]. In the case of HTTP booting, the file name must be set to `bootfile-name` and the value of this key must be the URL to the desired boot file [12]. In this project, the option is set to the URL of the GRUB NBP EFI file. For an example of how the configuration can look see Listing 1.

### 3.2.5 The Orchestration Service

To streamline communication between the orchestration service and other system components, a REST API is implemented, written using Go's net/http package in the stand-

```
{
        "state": "installing_os",
        "ip-address": "192.168.1.8",
        "timestamp": "2023-04-29T12:48:23.332805522+02:00",
        "message": ""
}
```

Listing 2: Example of a status on the status endpoint.

ard library[12]. This section will not go in-depth into the implementation on a code level. The REST API facilitates interaction between computers and users by utilizing JSON as a machine- and human-readable output format. As an established, widely adopted, and standardized approach to creating APIs, the REST framework offers several advantages [27]. It is lightweight and stateless, ensuring efficient communication and promoting ease of use.

The implemented API offers two endpoints: `client` and `status`. The `client` endpoint primarily facilitates the registration of new clients, which is initiated by the clients as defined in the Kickstart file. This endpoint expects the configuration file location to be sent during registration and returns an UUID for the client. Additionally, the `client` endpoint enables the setting of previously registered clients to an inactive state. This functionality is mostly useful for testing purposes, as the service automatically deactivates the old client if a client registers twice. Moreover, the endpoint allows for the retrieval of information about a specific client, or all registered clients. This information includes states, timestamps, active status, and the IP address.

The `status` endpoint permits the updating of existing clients' states. By providing the UUID and a new state, the client's information is updated within the system, along with any message sent with it. This endpoint also facilitates the retrieval of the newest state of a given client (see Listing 2). For an overview of the endpoints, refer to Figure 13.

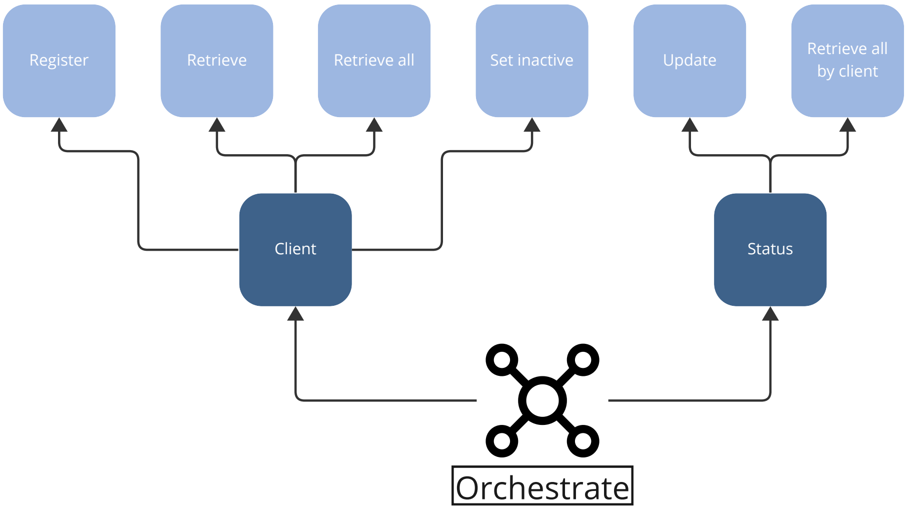


Figure 13: The orchestration service endpoints and their functionality.

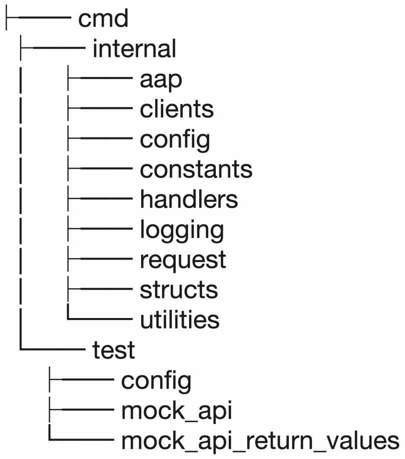
The service is using a configuration file, for two primary reasons. First, certain environment-dependent variables cannot be predetermined, such as the URL of the AAP instance, the organization ID in AAP, and the AAP-token that permits the creation of hosts and jobs. Second, to accommodate different environment specifications and requirements, the service allows for fine-tuning through optional variables. This flexibility ensures that the service can be deployed in various contexts where default values may not be suitable.

[12] https://pkg.go.dev/net/http, visited 20.05.2023

```
{
  "id": "abf23f84-46aa-4ba4-9a1f-c8a3a1ecb52f",
  "script-location": "test.sh",
  "status": [
    {
      "state": "registered",
      "ip-address": "[2.2.2.2]",
      "timestamp": "2023-04-22T17:56:12.9510397+02:00",
      "message": ""
    }
  ],
  "active": true
}
```

Listing 3: Example of a response from the client endpoint.

Examples of configurable variables include listening port, proxy URL, logging settings, and paths for TLS certificates and keys.



```
├── cmd
 ├── internal
 │   ├── aap
 │   ├── clients
 │   ├── config
 │   ├── constants
 │   ├── handlers
 │   ├── logging
 │   ├── request
 │   ├── structs
 │   └── utilities
 └── test
     ├── config
     ├── mock_api
     └── mock_api_return_values
```

Figure 14: Overview of the package structure in the orchestration service.

The service consists of multiple independent Go packages (see Figure 14), which each has its own responsibilities, following the concepts of cohesion and coupling. The packages in the service are thoroughly tested, more information on this is in Section 6.3. The different implemented packages in the service include:

**The clients package** is responsible for creating, storing, and updating the clients that are provisioned, together with their statuses and metadata (see Listing 3).

**The request package** is a utility package that offers functionality for sending HTTP-requests, including encoding of the request body. It also facilitates adding any necessary header to each request.

**The config package** reads and exposes the user-provided configuration variables for the rest of the application. By keeping this functionality restricted in its own package, it is ensured that other parts of the service cannot alter the variables after they have been initialized. When creating this package, it was important to support adding new variables in the future. Therefore, each part of the package is built up by modules, where the different functions that read and create each variable are loosely coupled to each other.

**The logging package** creates the necessary logging functionality that is used in the rest of the application. The logs are printed to the console and to a log file. The log file location is set by a configuration variable. The package utilizes zap[13], which implements multiple levels of logging: `DEBUG`, `INFO`, `WARNING`, `ERROR` and `FATAL`. The level is set in the configuration and makes it possible to customize how comprehensive the logs should be.

**The aap package** handles all communication with the Ansible Automation Platform (AAP) instance through its REST API. This includes:

- using the `AAP_TOKEN` environment variable to authenticate with AAP
- creating the client as a host in AAP
- creating an AAP inventory with the host
- initializing AAP jobs against a host
- checking if the status of the jobs against a host

Moreover, the package needs the configuration variables `aap_url` and `aap_organization_id` to be set, in addition to supporting an optional custom inventory prefix through the `aap_inventory_prefix` variable.

**The handlers package** implements the REST API endpoints, `/api/v1/status/` and `/api/v1/client/` (see Figure 13). These handlers act as the entry points for the rest of the packages.

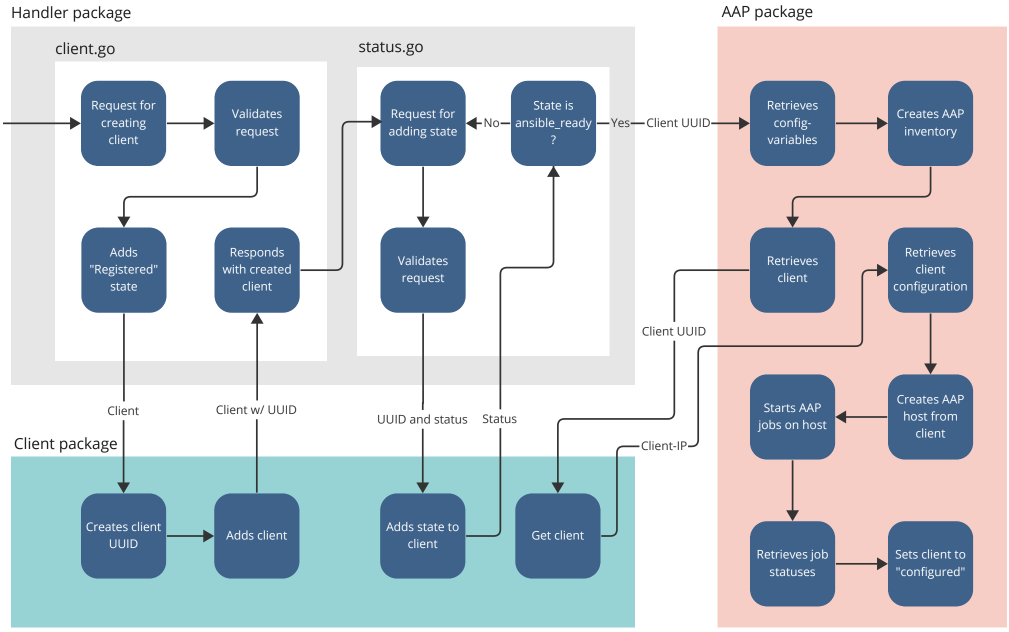
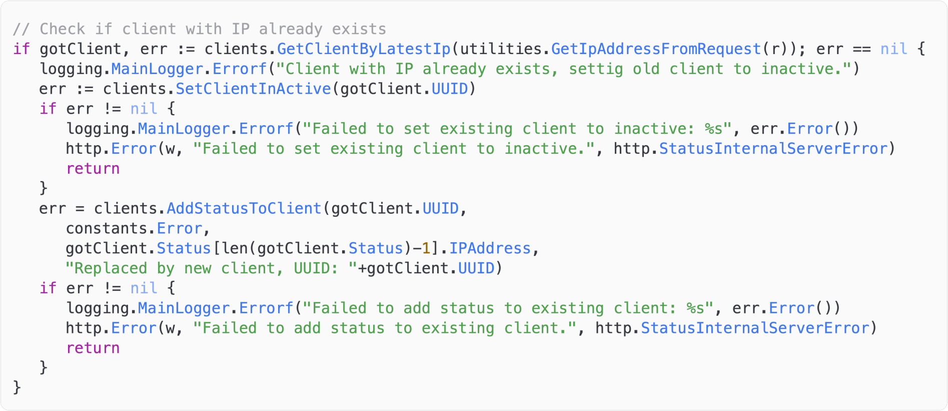


Figure 15: Overview of flows in the orchestration service throughout a successful provision.

In Figure 15, the flow of actions within the service during the successful provisioning of a single client is illustrated. In cases where the provisioning is unsuccessful, errors are documented in both the console and the log file. Moreover, the affected client will be assigned an error state, accompanied by a message describing the cause of the failure. However, clients may not always experience provisioning failures that can be communicated to the orchestration service. To accommodate such clients, they are allowed to register again. In such instances, the previous client is set as inactive, enabling the new

[13] https://pkg.go.dev/go.uber.org/zap, visited 20.05.2023.

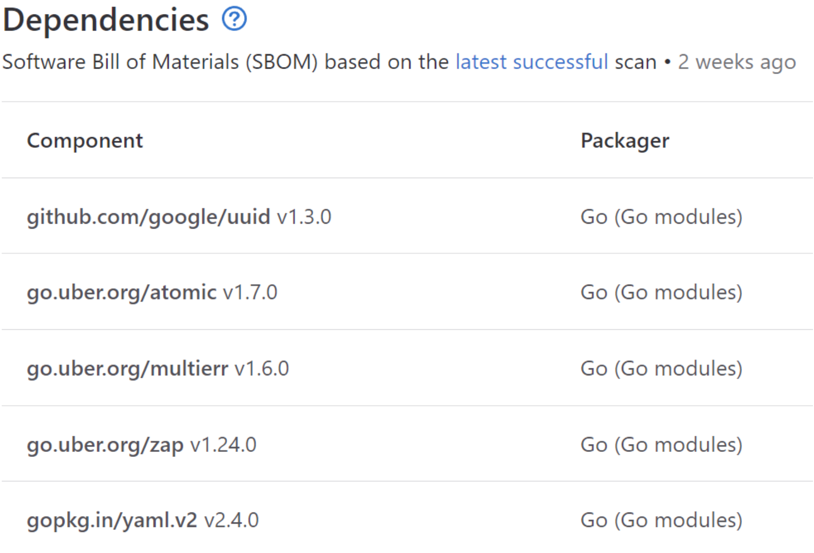
client to proceed with its provisioning process. Two clients are considered identical if they register with an IP address that is already present within the service. The code enabling this is shown in Figure 16.



```go
// Check if client with IP already exists
if gotClient, err := clients.GetClientByLatestIp(utilities.GetIpAddressFromRequest(r)); err == nil {
   logging.MainLogger.Errorf("Client with IP already exists, settig old client to inactive.")
   err := clients.SetClientInActive(gotClient.UUID)
   if err != nil {
      logging.MainLogger.Errorf("Failed to set existing client to inactive: %s", err.Error())
      http.Error(w, "Failed to set existing client to inactive.", http.StatusInternalServerError)
      return
   }
   err = clients.AddStatusToClient(gotClient.UUID,
      constants.Error,
      gotClient.Status[len(gotClient.Status)-1].IPAddress,
      "Replaced by new client, UUID: "+gotClient.UUID)
   if err != nil {
      logging.MainLogger.Errorf("Failed to add status to existing client: %s", err.Error())
      http.Error(w, "Failed to add status to existing client.", http.StatusInternalServerError)
      return
   }
}
```

Figure 16: Code for adding a client with an IP address that already exists.

When implementing the orchestration service, several third-party packages were utilized. These packages efficiently addressed some common tasks which would have been impractical for the group to develop. One notable example is the ’zap’ logging package. This package not only facilitated logging to both file and console but also offered log levels, a functionality absent from the standard Go logging library. Furthermore, packages for generating UUIDs and parsing YAML were employed. Utilizing an established YAML file parser contributes to the overall security of the application, predicated on the assumption that an existing, widely-used package would be more thoroughly tested than a self-made solution. For a comprehensive list of all dependencies employed in this project, including both direct dependencies and their subsequent dependencies, refer to Figure 17.



## Dependencies ?

Software Bill of Materials (SBOM) based on the latest successful scan • 2 weeks ago

| Component | Packager |
|---|---|
| github.com/google/uuid v1.3.0 | Go (Go modules) |
| go.uber.org/atomic v1.7.0 | Go (Go modules) |
| go.uber.org/multierr v1.6.0 | Go (Go modules) |
| go.uber.org/zap v1.24.0 | Go (Go modules) |
| gopkg.in/yaml.v2 v2.4.0 | Go (Go modules) |

Figure 17: List of dependencies for the orchestration service, fetched from the GitLab dependency list.

### 3.2.6 Kickstart Communication

The client undergoing the provisioning process must register itself to the orchestration service, in addition to communicating its progress. This is done through the Kickstart file's PRE and POST script sections. The PRE scripts are executed before the OS installation on the client, while the POST scripts are run after the OS installation, but before the client reboots.

These scripts are utilized for the necessary communication with the orchestration service. By defining CURL-requests within these sections, the client registers and sends the state `installing_os` in the PRE section, and `installed_os`, `rebooted`, and `ansible_ready` in the POST section (see Figure 18). However, this approach encountered several challenges.

```
curl -kX POST -H "Content-Type: application/json" -d '{
                                        "client-id": "'"$id"'",
                                        "state": "ansible_ready"
}' https://ORCHESTRATOR_IP/api/v1/status/
```

Figure 18: Example of a CURL-request.

The first challenge was how the client would maintain its UUID throughout the process in order to accurately update its state. This is addressed by persisting the UUID, which is returned upon registration, to a file on the client's file system in the install environment. This approach enables other CURL-requests, executed in the different sections, to access the content of the file and utilize it to identify itself to the orchestration service. To facilitate this process, the `only_id` URL parameter is implemented at the client endpoint, ensuring that the response body exclusively contains the UUID string.

By default, the POST section of a RHEL Kickstart operates within a chroot-jail[14]. In this mode, it is unable to access the same files as the PRE section. Consequently, the POST section is invoked with the `--nochroot` parameter to guarantee access to the UUID file.

Since the POST section is executed prior to the reboot, it is not the appropriate location to send `rebooted` and `ansible_ready` states. This challenge is addressed within the POST section. The appropriate communication with the orchestration service is achieved by developing a script containing the necessary CURL-requests and establishing a systemd service that executes the script following a reboot. A complete example of a Kickstart file including the PRE and POST sections can be found in Appendix V.

## 3.3 Project and Development Process

The project plan (see Appendix K) details the approach the group took to execute the project, including the development model used and how the group would adapt the framework to meet their specific needs. This section builds upon this plan by providing an overview of how it was implemented, explaining the degree to which it was followed, and providing a summary of the execution. It will not include a description of the group's experience using this model. Please refer to Section 7.7 for this discussion.

[14] A security mechanism in Unix-like operating systems that isolate a process and its subprocesses within a restricted file system. Read more at https://phoenixnap.com/kb/chroot-jail, visited 20.05.2023.

### 3.3.1 Development Model

The group decided that a combination of Scrum and Kanban would serve as the development model for this project. Further elaboration and justification for this choice can be found in Appendix K. The Kanban board provided a visual workflow which gave a clear overview of the tasks, referred to as ”issues,” that needed to be completed throughout the project. On the other hand, Scrum provided a more structured approach to the project by breaking it down into sprints for more organized progress. The group combined the Kanban board with Scrum sprints.

During the planning phase, the group created a Gantt diagram that outlined the expected progression of the project, as detailed in the project plan (refer to Appendix K). However, the initial diagram did not accurately reflect the project’s progress. Due to unforeseen workload outside this project, the group created a revised Gantt diagram in the middle of the project’s timeline, to provide a more realistic overview of how the progress was and would be, as shown in Figure 19. For a larger version of the diagram see Appendix T.

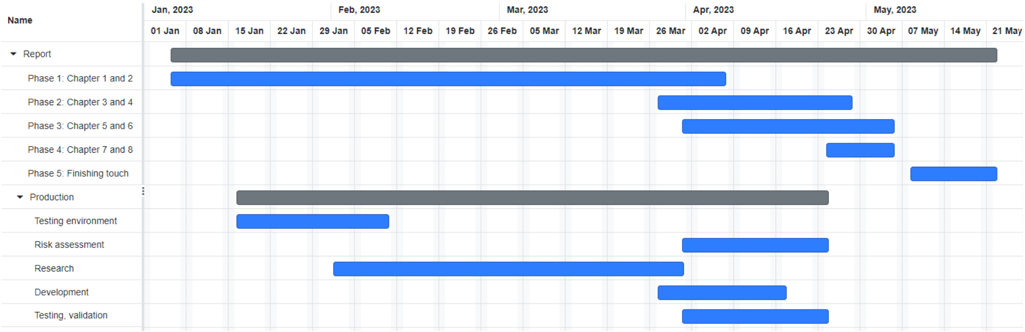
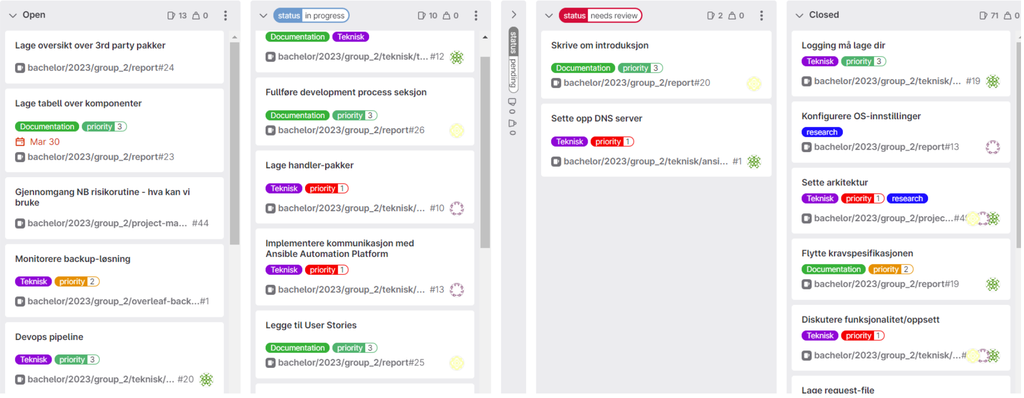


Figure 19: Revised Gantt diagram.

**Kanban Board**

The Kanban board served as a visual tool for managing the workflow of a project by categorizing, prioritizing, and limiting work in progress [42]. To achieve this, the workload was divided into separate tasks. Each task was represented as an issue in GitLab. The issues were labelled to indicate their respective categories and assigned a state that changed as the issue was worked on and moved around the board. The board provided an overview of the state of each task, who it was assigned to, and its priority. This ensured that multiple members did not work on the same issue and that no issue was neglected. The progress of issues was documented by commenting on the issue when there was an update. This kept all information and discussion relevant to the issue in one place and enabled all group members to stay informed about the progress of the issue.

The board was composed of columns that represented the different states an issue could be in; *Open, In Progress, Pending, Needs Review, and Closed.* The Pending state indicated that an issue was awaiting further action before being continued, such as another issue or a confirmation from the employer. All issues were reviewed by another group member before being closed to ensure quality. To make it easier to identify the characteristics of each issue, such as urgency and importance, the labels, priority levels, and statuses were colour-coded. This visual cue helped team members quickly and easily recognize which part of the project the issue belonged to. A snippet from the Kanban board can be found in Figure 20.



Open 13 0

Lage oversikt over 3rd party pakker
bachelor/2023/group_2/report#24

Lage tabell over komponenter
Documentation priority 3
Mar 30
bachelor/2023/group_2/report#23

Gjennomgang NB risikorutine - hva kan vi bruke
bachelor/2023/group_2/project-ma... #44

Monitorere backup-løsning
Teknisk priority 2
bachelor/2023/group_2/overleaf-back...#1

Devops pipeline
Teknisk priority 3
bachelor/2023/group_2/teknisk/... #20

status in progress 10 0

Documentation Teknisk
bachelor/2023/group_2/teknisk/t... #12

Fullføre development process seksjon
Documentation priority 3
bachelor/2023/group_2/report#26

Lage handler-pakker
Teknisk priority 1
bachelor/2023/group_2/teknisk/... #10

Implementere kommunikasjon med Ansible Automation Platform
Teknisk priority 1
bachelor/2023/group_2/teknisk/... #13

Legge til User Stories
Documentation priority 3
bachelor/2023/group_2/report#25

status pending 0 0

status needs review 2 0

Skrive om introduksjon
Documentation priority 3
bachelor/2023/group_2/report#20

Sette opp DNS server
Teknisk priority 1
bachelor/2023/group_2/teknisk/ansi... #1

Closed 71 0

Logging må lage dir
Teknisk priority 3
bachelor/2023/group_2/teknisk/... #19

Konfigurere OS-innstillinger
research
bachelor/2023/group_2/report#13

Sette arkitektur
Teknisk priority 1 research
bachelor/2023/group_2/projec...#4

Flytte kravspesifikasjonen
Documentation priority 2
bachelor/2023/group_2/report#19

Diskutere funksjonalitet/oppsett
Teknisk priority 1
bachelor/2023/group_2/teknisk/...#

Lage request-file

Figure 20: Snippet of Kanban board during Sprint 4.

### Scrum Sprints

By utilizing the Scrum framework, the project adopted a structured approach to the project. The project timeline was divided into sprints with an initial duration of 10 business days each. However, due to unforeseen events, the group adopted the duration of each sprint as needed to ensure successful completion. The sprints were represented as *Milestones* in GitLab[15], with issues linked to them. This allowed the group to prioritise and address issues related to the sprint goal while ensuring that other issues were not forgotten and could be addressed at a later time.

At the start of each sprint, the group held a sprint planning meeting. During this meeting, the group reviewed the still open issues from the previous sprint and decided whether to close them or move them to the new sprint. The group also established a sprint goal and identified the work needed to achieve it by assessing progress on the Gantt diagram and incorporating feedback from the employer and academic supervisor. Based on the assessment, broad issues were avoided and specific and concrete issues were created instead. As the sprint progressed, the group created additional issues as needed to meet the sprint goal. At the end of each sprint, the group held a sprint retrospective meeting. During this meeting, the group evaluated the effectiveness of the sprint, identified areas of improvement for future sprints, and addressed any remaining issues or concerns.

In addition, the group held daily status meetings to ensure everyone was aware of the progress of the sprint and its issues. These meetings focused on discussing the work done the previous day, plans for the current day, and obstacles encountered, if any. These brief meetings allowed all group members to stay informed about the sprint's progress and any changes that needed to be made to meet the sprint goal, such as prioritizing issues depending on the progress.

## 3.3.2 Summary of Sprints

This section will summarise the sprints in this project. The project was divided into one planning phase and six sprints, during which the group developed the provisioning system and completed the thesis.

### Planning phase (Jan 4th to Jan 20th)

During the initial stage of the project, the group spent several weeks creating a project plan. This involved meetings with the employer and academic supervisor to discuss their respective expectations, both in terms of technical aspects and documentation. The group also discussed and agreed upon the expected progress within the group.

[15] https://docs.gitlab.com/ee/user/project/milestones/, visited 19.05.2023

**Sprint 1 (Jan 23rd to Feb 3rd)**
The first sprint focused on completing the project plan, in addition to creating a project contract. The group also drafted an outline of the report based on input from the academic supervisor. Furthermore, the group configured a testing environment at the NTNU campus.

**Sprint 2 (Feb 6th to Feb 17th)**
In the second sprint, the group had meetings at the employer to get an understanding of their security assessment procedures (see Section 3.3.3). The group spent the early part of the sprint preparing for these meetings. Furthermore, the group completed the deployment of the test environment, began working on the first section of the bachelor thesis, and initiated the planning and research of various system components, such as network booting, Kickstart files, and automatic OS configuration.

**Sprint 3 (Feb 23rd to Mar 3rd)**
During the third sprint, the group did not add any new issues and instead concentrated on completing the remaining research from the previous sprint. The group encountered unforeseen challenges during this sprint, including unexpected time-consuming factors and sickness among several group members. As a result, the research issues were not completed and had to be carried over to the following sprint as well.

**Sprint 4 (Mar 23rd to Apr 11th)**
The fourth sprint was the start of the project's technical phase. The goal for this sprint was to establish the solution's architecture with guidance from the employer and to consequently close the remaining research issues carried over from the previous sprint. The sprint kicked off the development of the solution. Additionally, the group made necessary adjustments to the thesis report structure and began writing the first sections.

**Sprint 5 (Apr 12th to Apr 28th)**
During the fifth sprint, the group finished the development of the solution, with a hard deadline at the end of the sprint set by the group. This included the implementation of the orchestration service, Ansible roles and the integration with AAP, as well combining the components to a seamless provisioning system. The system was tested at the employer to ensure that the needed functionality was implemented. Furthermore, the group continued writing the thesis.

**Sprint 6 (May 1st to May 22nd)**
The sixth sprint spanned over the last three weeks of the project and was dedicated to documentation and quality assurance of the solution. Early in this sprint, the group conducted a comprehensive security assessment with guidance from a Security Architect (SARC) from the employer. In addition, the group conducted user tests of the solution. The primary focus of this sprint, however, was to write and finalize the thesis for submission by the May 22nd deadline.

### 3.3.3 Meetings

Throughout the project, the group held various types of meetings on a regular basis. These were meetings with the employer and the academic supervisor, as well as internal meetings. The meetings served the purpose of keeping all relevant parties informed of the project's progress and provided a platform for feedback and assistance when necessary. In addition, the group occasionally held meetings with employees at the employer for professional and educational purposes. Meetings were primarily conducted on the digital platform Teams [67], but also at the employer, while internal meetings were mostly held at campus. See Appendix M for examples of summaries of these meetings.

**Employer**
The group held weekly meetings with the employer every Friday at 11 am. During these meetings, the group provided updates on their progress since the previous meeting, addressed any questions or concerns that may have arisen, and discussed plans for the upcoming week. These meetings were mainly focused on the technical aspect of the project.

**Academic supervisor**
The group held weekly meetings with the academic supervisor every Friday at 12 pm. During these meetings, the group provided updates on their progress since the previous meeting, addressed any questions or concerns that may have arisen, and discussed plans for the upcoming week. These meetings were mostly focused on thesis writing and other academic aspects.

**Internal**
The group conducted daily status meetings from Monday to Friday to ensure regular progress updates. These meetings were focused and brief. Aside from the daily status meetings, the group held sprint planning and retrospective meetings, before and after each sprint respectively, which were more extensive and allowed for a comprehensive understanding of the sprint's goal, as well as identifying key takeaways to improve the next sprint.

# 4 Deployment

This section provides a comprehensive overview of the considerations necessary for deploying the provisioning system. It covers the deployment requirements, the recommended deployment environments, and the compatible client operating systems. In addition, it details the processes for monitoring, maintaining, and scaling the system. The section also describes how the provisioning system is deployed within the employer's environment. The deployment and configuration of Ansible Automation Platform (AAP) are out of scope of this project. The deployment of components, such as the ones in this solution, can require a considerable amount of time and resources. However, in this solution, all components are deployed and configured using Ansible. This significantly simplifies the deployment procedures.

## 4.1 Deployment Requirements

The provisioning system has some requirements that need to be met before it can be deployed. Ensuring that these are fulfilled is outside the scope of this project:

- The base RHEL 9 Operating System on the server(s) must be installed and configured.
- The network must be configured.
- An OS package repository with the required packages must be available.
- An Ansible Automation Platform (AAP) instance must be available.

The AAP instance must be deployed in such a way that it is reachable over HTTP by the orchestration service, and that it can reach the clients over SSH. It must also have been configured with the correct machine credentials, an organization initialized, and job templates to accommodate the provisioning system.

As a consequence of the specification in Section 2.1.1, the system is primarily designed to work with Dell clients. This does however not necessarily exclude the compatibility of other client vendors, but this has not been verified.

## 4.2 Supported Client Operating Systems

The only tested and verified Linux distribution is RHEL 9. In theory, this system should be able to provision any Operating System. However, since the GRUB menu creation currently specifies some Red Hat family distribution kernel parameters, some modifications must be done in order to, for example, provision a system with Debian. Moreover, the Operating System needs to be supported by Ansible.

## 4.3 Deployment Environment

The provisioning system aims to be deployed on a virtual machine as specified in Section 2.1. This does however not exclude a system administrator from deploying the system on a bare-metal server or into a containerized environment, but the system is not created with this in mind. The Ansible playbook aims to be flexible in terms of deployment. All components can be deployed on the same or separate machines and all, but the orchestration service, can be deployed in multiple replicas. As a result, the deployment environments might vary greatly. This also makes it easier to deploy multiple instances of the systems in different networks using the same Ansible playbook. Having different hosts for each component lets the system administrator have a greater separation of duties on the host machines.

To limit people's access to provision clients, the provisioning system should be on a separate Virtual Local Area Network (VLAN). This ensures that the provisioning network is separated from the rest of the network. Ideally, the VLAN should only be available on specific physical ports in a secure area of the building.

In addition to having the system on a separate VLAN, this VLAN should have limited internet access and limited access to internal services that are not needed during the provisioning process (as illustrated in Figure 21). This can be accomplished by using a proxy or blocking the internet access completely. However, having limited internet access might affect the ability to install packages and other tasks that typically requires internet access, hence, the Ansible playbooks for client configuration might need to be modified. In addition, the OS ISO-image need to be self-contained, and it is, therefore, essential to ensure that all necessary files and packages are included within the OS ISO-image. However, this inclusion requires additional maintenance, as outlined in Section 4.5.

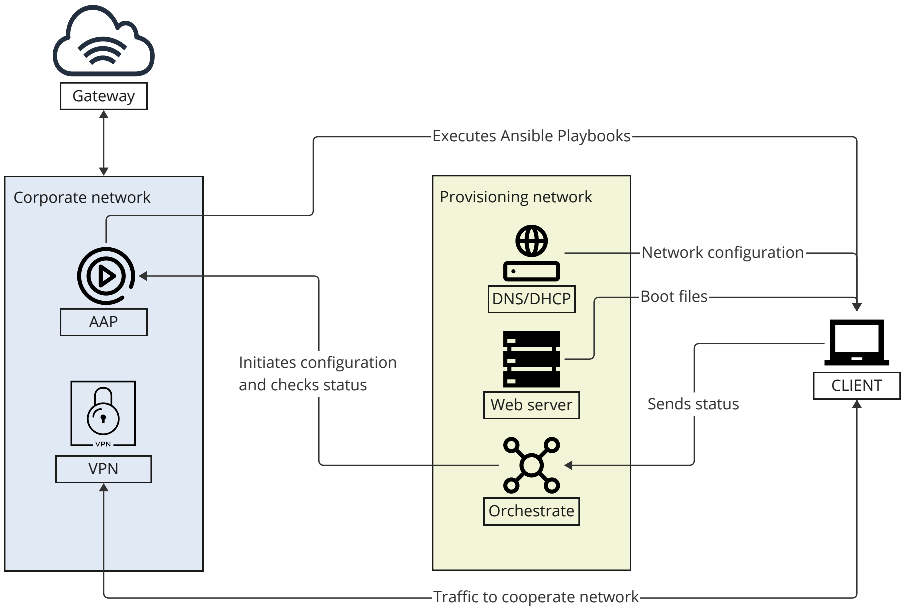


Figure 21: Illustrates the network bound and communication flows in an air-gapped deployment environment.

## 4.4 Deployment of the System

In theory, all components of the provisioning system can be deployed using the method the administrator desires. However, the preferred method is utilizing the Ansible playbook to deploy and configure the system (see Section 3.2.1). The simplest and most recommended method of deploying the system is to use a single host. In most cases, this should be proficient enough to handle the simultaneous installation of several clients. The Ansible installation playbook includes the additional settings to the OS that are needed for the provisioning system to work. The Ansible playbook has only been tested on RHEL 9 as specified in the functional specifications (see Section 2.1.1), therefore, this is the recommended Operating System to host the system. The other Linux distributions of the Red Hat family should work as well with no or a few modifications. For the complete steps for the deployment of the system see Appendix P.

## 4.5 Monitoring and Maintenance

In regards to the provisioning system itself, there should be little maintenance necessary. An exception is updating the software packages when needed, such as NGINX. This can be combined with an existing patching routine of the OS since the additional packages are installed through the default RHEL package manager. In some cases, there will be breaking changes in the software that necessitates a configuration change in the Ansible roles. In the future, some of the configuration options might no longer be deemed secure and should then be changed to reflect this.

Regarding the orchestration service, there should be very few maintenance tasks. However, one vital part is to update third-party libraries when vulnerabilities are discovered and patches are created. This might necessitate changes in the code if parts of the library introduce breaking changes. The process of updating third-party libraries is more involved than updating system packages since the service must be recompiled. This part is currently handled by the GitLab pipeline, making the task easier (see Section 6.4 for more information).

Another aspect of the maintenance is the client configurations. This consists of Operating System ISO-image, Kickstart files and Ansible playbooks in AAP. When updated versions of an OS are released, the OS ISO-image on the HTTP file server should be updated, to make sure the most up-to-date version is present. If the provisioning system is deployed in an environment with no internet access, the OS ISO-image should be updated often to make sure that the most recent security patches are initially applied. Updates in the OS ISO-image might necessitate changes to the Kickstart file as well. The Ansible playbooks used to configure the client will probably contain frequent changes and it will be necessary to have a simple and easy way to update the projects and job templates in AAP to reflect these changes.

For monitoring, the status endpoint can be utilized manually to view the states of the provisioning jobs in the memory of the orchestration service application. This endpoint provides a simple overview of clients, including installation timestamps, completed stages, and any potential failures. Additionally, logs from the orchestration service, NGINX, dnsmasq, AAP and the client itself can be analysed. Here, the system administrator can closely monitor system activities. The logs enable the possibility to utilize a log analysing tool, to look for anomalies or create statistics based on the log entries. Furthermore, the network traffic can be captured to gain insights into the flow of the data between the system components. This can help look for bottlenecks or malicious packets on the network.

## 4.6 Scalability

The Ansible playbook which deploys the provisioning system has the ability to deploy the components of the provisioning system on different hosts. This makes it possible to have differently vertically scaled hosts dependent on how much resources the component needs. However, all the components can be deployed on the same node, if the resource requirements allow it.

The stateless components of the system, such as the web server and the DNS server, can easily be scaled horizontally. For instance, the DHCP server can provide a list of DNS servers for the client to use. When it comes to the web server, a reverse proxy or another form of load balancer must be placed in front to route the traffic between instances in a desired manner. A simpler load-balancing approach involves assigning a set of servers to specific ranges of physical Ethernet ports and another set for other ranges. The orchestration service can also be load balanced; however, each client must use the same instance throughout the entire installation process.

Vertical scaling[16] of the system is the easiest way of improving the performance of the system when provisioning many clients simultaneously. The horizontal scaling requires additional components like load balancers and reverse proxies, and should therefore only be considered for large-scale deployments systems.

## 4.7 Deployment at the Employer

This section has been redacted, refer to Appendix F.

[16] Increasing the resources available for each component.

# 5 Security Assessment

The automatic provisioning system, referred to as the system in this section, will be deployed in an environment with low risk acceptance. Therefore, one of the task specifications is to security assess the system (see Section 2.1.2). This includes considering the business impact in the case of a security breach, assessing threats, and identifying the vulnerabilities associated with the system and environment. Combined, these elements are the factors when defining and evaluating the security risks associated with the system. The security assessment consists of the following:

1. Business Impact Analysis (BIA)
2. Threat Assessment
3. Security Risk Assessment, including a vulnerability assessment

The security assessment is conducted in compliance with the framework provided by the employer. By doing so, the assessment is built upon a solid foundation, and the group is able to benefit from the employer's expertise and experience in the field. Additionally, using the existing framework help ensure that the assessment process meets industry best practices and complies with relevant standards and regulations, which potentially can increase the credibility of the assessment. Given the employer, it is important to ensure the confidentiality of sensitive information. Hence, the BIA can be found in Appendix B and C, the threat assessment can be found in Appendix D and the full security risk assessment, including the vulnerability assessment, can be found in Appendix E. Due to the confidentiality clause, these analyses will not be disclosed to the public.

## 5.1 Scoping

The scope is limited to the factors introduced by the provisioning system within the employer's environment. This environment is a secure and closed network, already in place and utilized by the employer. The assessment aims to identify and address potential threats, vulnerabilities and risks that may emerge from its integration within the existing environment. The components introduced are an NGINX web server, DNS server and DHCP server. Furthermore, the system introduces an orchestration service to the environment. Refer to Figure 8 for an overview of the high-level system architecture. The specifications of the BIOS and OS configuration, the setup of the OS on the server, and the process performed by AAP are out of scope of this assessment. Further description of the scope can be found in Appendix E.

## 5.2 Business Impact Analysis (BIA)

A Business Impact Analysis (BIA) aims to predict the impact of a disruption in a system or a business function. In addition, the BIA tries to collect the necessary information needed to develop a recovery strategy tailored to the specific function or to the business as a whole. The analysis should include aspects such as revenue and income, delays in business services, increased expenses, fines, and repercussions on customer reputation. By evaluating these aspects, businesses can prioritize the restoration of services in the event of downtime [6] [68]. In the context of IT and cybersecurity, BIA aims to identify the IT components associated with business processes and assess the worst-case consequences of unwanted actions from threat actors. Through this information, a better IT disaster recovery plan, business recovery plan, and incident management plan can be developed [10].

Involving relevant stakeholders when conducting a BIA, facilitates the identification of any inherited aspects from other systems. For instance, if a host system of a service has a minimum up-time of 98%, the service cannot maintain a minimum up-time of 99%. Other up-time requirements, such as Maximum Acceptable Outage (MAO), Recovery Time Objective (RTO), and Recovery Point Objective (RPO), should also be established for the system [47].

The background for the BIA and its justifications can be found in Appendix B, while the BIA report can be found in Appendix C.

## 5.3 Threat Assessment

The threat assessment is an essential part of a risk assessment. It involves evaluating potential actors who may pose a threat to a system, along with how they might cause harm and their ability and motivation to do so. First, it is important to gain an overview of the different attack vectors of the system. Next, it is necessary to identify potential threat actors, as well as their motivation, capability, and capacity to exploit these attack vectors. Details of this assessment can be found in Appendix D.

## 5.4 Security Risk Assessment

The security risk assessment combines assessments and considerations to define and evaluate the various risks associated with services and systems. The assessment is conducted by combining the result of the BIA and threat assessment, as well as a vulnerability assessment, to calculate a risk score and its respective risk level. The vulnerability assessment is performed by evaluating the different attack vectors from the threat assessment in relation to the weaknesses of the system and is given an exposure level based on the difficulty of exploiting said weakness [57].

The assessment suggests risk treatment of the risks that are too severe, ensuring that the total resulting risk is below the risk appetite [57]. The assessment can be found in Appendix E.

## 5.5 Appendices Overview

The appendices related to the security assessment are the following:

- For the BIA, see Appendix B and C
- For the threat assessment, see Appendix D
- For the Security Risk Assessment, see Appendix E

# 6 Quality Assurance

This section discusses the various measures employed by the group to ensure high-quality and effective collaboration throughout the project. The section is divided into five parts. The **first** part addresses the quality assurance process from the employer, which aimed to ensure quality and security to the entire software development process of the project. The **second** part focuses on individual measures taken to maintain consistency and quality during development, covering aspects such as coding style and practices, functionality, and security. The **third** part discusses the measures introduced for the evaluation of each other's work and the promotion of sufficient collaboration. The **fourth** part covers additional measures implemented which are not directly part of the development process itself but were introduced to ensure the robustness and security of parts of the system. In the **fifth** part, the section addresses various quality assurance measures involving professional parties. In general, all parts describe the different measures the group employed to deliver a solution that meets the industry's best practices and the employer's needs, including security, reliability, and overall quality.

## 6.1 Secure System Development Life Cycle (SSDLC)

To ensure the quality and security of the system's life cycle, the SSDLC of the employer was utilized. The completed SSDLC checklist can be found in Appendix H. Having an SSDLC is important to ensure that security is considered in the earlier stages of the development, as well as in every step of the process. This approach mitigates the risk of making security an afterthought and discovering vulnerabilities in the later stages of the system life cycle. Resolving a security issue earlier in the system life cycle proves more cost-effective for an organization [65].

The relevant steps of the SSDLC were followed during the project, however, the checklist was not necessarily completed in the specified sequence. In addition, the security requirements are not necessarily as detailed as desired by the SSDLC. Using the SSDLC, made sure that security was made an integral part of most choices during the planning, implementation and assessment stages of the project. The SSDLC describes steps that should be conducted during the concept, planning, implementation and handover phase. Most stages aim to assess and ensure that risks are handled appropriately and that the concept is well-defined with a set of security requirements. The most important stage this project incorporated was the business impact, threat and risk assessment stage (see Section 5 for the assessments). These assessments helped find potential security issues and find remediation steps. In addition, they work as a way to ensure that the incorporated security measures had an effect.

## 6.2 Code Standards and Practices

Following code standards and practices are crucial because they ensure consistent and maintainable code, making it easier for the group to understand and collaborate. Additionally, they help prevent errors and enhance code efficiency, leading to higher-quality software and a smoother development process.

The group focused on providing sufficient comments within the codebase, particularly when the code was not self-explanatory. In addition, the group aspired for loose coupling among components and minimise dependencies, making the software more adaptable to future changes. To promote usability, the group has created a README that outlines essential information, such as variables and API functionality. Furthermore, user manuals and other documentation were written in the markdown format, this being the preference of the employer, and to ensure ease of access and readability (see Appendix P and Q). The group followed standards and best practices, including following the package structure standards defined by the Go community[17] (see Figure 14), using the appropriate HTTP status codes[18] and using camelCase and PascalCase[19] for the naming convention. Resulting in a system that aims to make it easy for future users and developers to understand and maintain the software.

## 6.3 Testing

Testing is a fundamental aspect of software development that ensures the reliability, functionality, and security of a software product. This section will discuss the importance of testing during the development process and how it was employed in the project. The testing was conducted on the orchestration service, and on the automatic provisioning system as a whole.

One of the primary reasons testing is crucial in software development is its ability to identify potential vulnerabilities. As security breaches can result in severe consequences, it is essential to address these issues during the development process. By conducting thorough testing, developers can detect and mitigate risks, ensuring that the final product is secure and robust. Furthermore, testing ensures that the software performs as expected. It can help developers identify and fix discrepancies between the desired and actual behaviour of the software.

The testing methodology utilized encompassed both positive and negative test cases. Positive tests are designed to verify that the software functions as expected under normal conditions, while negative tests assess its behaviour under erroneous or unexpected conditions. This approach ensured that the software was resilient and adaptable in the face of a wide range of scenarios. The group also adhered to the best practice of continuous testing by testing each new feature in the code during the development process. This approach allowed the group to identify and address issues early in the development process, reducing the likelihood of discovering critical bugs later on. By integrating testing into the development workflow, the group was able to maintain a high level of quality and ensure that the software met the desired specifications.

### 6.3.1 Unit Testing

The group used unit testing to evaluate the functionalities of the orchestration service. Unit testing involves breaking down a program into discrete, testable behaviours and assessing them individually as units. By integrating continuous testing with unit testing, each functionality was tested to ensure it performed as expected before proceeding to the next component of the service. Unit testing encouraged modular code design with a clear separation of functionality in the code, resulting in more maintainable and scalable software. Additionally, by testing individual components at the earliest stages of development, the time and effort needed for debugging and error correction were minimized. As a result, the overall efficiency of the development process was improved.

When writing unit tests, the group utilized mocks to substitute some of the functionalities of the complete code or external dependencies. This approach allowed the tests to concentrate solely on the business logic without being dependent on other modules. Furthermore, by focusing on the input and output of the mocked functions, the tests became easier to implement. For instance, the tests mocked the AAP functionalities. Utilizing mocks in this manner enabled a more controlled and isolated test environment, which in turn, contributed to more accurate and reliable test results.

The unit testing was primarily implemented using table-driven testing, which is a method

[17] https://github.com/golang-standards/project-layout, visited 16.05.2023.
[18] https://www.rfc-editor.org/rfc/rfc7231, visited 21.05.2023
[19] https://gyulgrigoryan.medium.com/variables-in-golang-1bf492df03b0, visited 20.05.2023.

```
Function TestAddClient(t)

  Define args struct containing a client
  Define tests array with name, args, and expectedError

  tests = [
    {
      name: "Add client without UUID",
      args: {
              client: {
                      Status: []
              }
            },
      expectedError: nil},
    {
      name: "Add client with UUID",
      args: {
              client: {
                        UUID: "3",
                        Status: []
                      }
            },
      expectedError: ErrClientIDGiven
    }
  ]

  For each test in tests
    t.Run(test.name, Function(t)
      result, err = AddClient(test.args.client)

      If err is not equal to test.expectedError
        t.Error("AddClient() error =", err, "expectedError =", test
           .expectedError)
      End If

    End Function)
  End loop

End Function
```

Listing 4: Psuedo code for table-driven testing

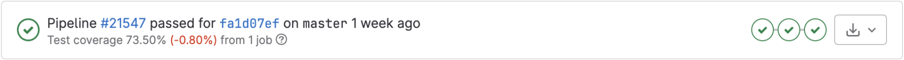
for conducting unit tests in Go. They were implemented using the *Go testing* package[20]. This approach gave test cases that were both concise and easily maintainable. In table-driven testing, multiple test scenarios are defined using a list of test structs and then iterated over within a single test function. Each test case typically contains a name describing the test, the unit's input values, and its expected outcome. This method of testing made it easy to add, modify or remove tests if necessary during the development process.

Listing 4 represents a pseudocode representation of a test function. First, an argument struct (args) is defined, which is the input value of the function being tested, in this case containing a client instance. Second, a test array is defined, which includes the name of the test, args, and the expected error code. Thirdly, a test array is created with two elements, one with valid args and one without. Lastly, the test function loops through the elements in the test array as input to the function being tested and then compares the expected error code with the actual one. Refer to Appendix O for an actual Go example.

[20] https://pkg.go.dev/testing, visited 20.05.2023

#### Test Coverage

The group aimed to achieve high test coverage by creating unit tests for most components of the software. This approach enabled validation of the functionality and reliability of individual units, particularly the most critical ones. The test coverage statistics are generated by the GitLab pipeline (see Figure 22). By the conclusion of the project, the test coverage achieved was 73.5%. This indicates that a significant majority of the orchestration service's code was successfully tested to validate its functionality, reliability and security. See Section 6.4 for more information about the Gitlab pipeline.



Pipeline #21547 passed for fa1d07ef on master 1 week ago
Test coverage 73.50% (-0.80%) from 1 job

Figure 22: The test coverage of the orchestration service, as shown in a merge request in GitLab.

### 6.3.2 Functional Testing

During the development of the Ansible playbooks and roles, functional tests were performed regularly to ensure that the scripts functioned as intended. This was done by manually executing the scripts on a test server. These tests could have been conducted automatically, but this would require a significant investment of time and was therefore not prioritised.

Manual functional tests were also conducted on the orchestration service throughout its development. This ensured that the different packages functioned together as intended and that the service as a whole met the architecture specifications.

### 6.3.3 System Testing

Some system tests were conducted at the employer to verify that the system was working as intended. The system tests could not be conducted in the local development and test environments due to the lack of access to clients that supported UEFI HTTP boot. In addition, testing in the test environment was not ideal since the employer's environment includes additional security hardening which could not easily be replicated in the test environment.

It was tested that the UEFI HTTP boot and the Kickstart file worked, before involving the orchestration service. This made it simpler to isolate issues before a complete system test was conducted. It would probably have been possible to automate these steps by deploying all needed servers through IaC and having a client that could be controlled remotely. However, this was deemed unnecessary for a project of this size given the time needed to implement this alternative.

Executing system tests resulted in a better understanding of how the provisioning system could be improved, in addition to verifying that the system worked as expected. This information was vital for the further development of the provisioning system. For instance, SELinux-related issues were found that would not otherwise have been discovered, as well as issues related to TLS cipher suites (see Section 7).

## 6.4 Continuous Integration (CI)

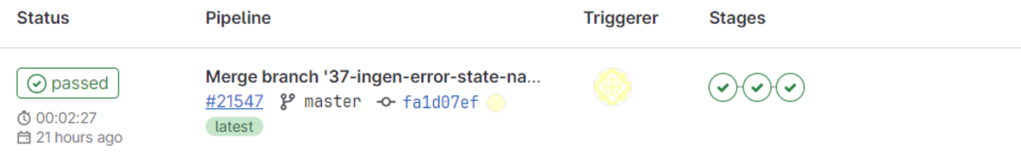
The group employed continuous integration through GitLab pipelines that were triggered automatically whenever new code was pushed to the project repository[21]. The pipeline for the orchestration service was composed of various stages, each performing specific

[21] Gitlabs solution for CI/CD. https://docs.gitlab.com/ee/ci/pipelines/, visited 20.05.2023.

tasks to ensure the code's quality and functionality. In contrast, the pipeline for Ansible focused on linting and style checks, ensuring that the code adheres to best practices and established conventions. Moreover, the code had to pass the pipeline implemented by the employer to ensure that it met their standards and expectations.

### 6.4.1 Orchestration Service Pipeline

The group implemented different stages for the orchestration service pipeline, which are integrated into the GitLab pipeline (see Figure 23). Firstly, a testing stage was implemented to automate the testing and make sure that all tests passed. Secondly, pipeline stages provided by GitLab that focus on application security, were implemented. Lastly, a build stage served to automate and maintain consistency in the application's build process (see Appendix R for the full pipeline implementation).



| Status | Pipeline | Triggerer | Stages |
|---|---|---|---|
| ✓ passed<br>00:02:27<br>21 hours ago | Merge branch '37-ingen-error-state-na...<br>#21547 master fa1d07ef<br>latest | | ✓✓✓ |

Figure 23: Example of a commit passing the pipeline stages.

**Test**
The test stage downloads the required Go modules and dependencies and runs the unit tests across all sub-packages. It also generates a coverage report (see Section 6.3.1). The stage automatically runs all tests for every push to the branch, and would not pass if any of these failed. This approach allowed the group to maintain a high degree of confidence in the code's stability, as all changes were examined before being incorporated into the master branch. This streamlined approach also reduced human error, such as merging non-functional code to the master branch. Furthermore, it enhanced efficiency and ensured consistent testing throughout the life cycle of the project.

**GitLab SAST**
Static Application Security Testing (SAST) is a testing methodology that analyzes source code to identify potential vulnerabilities, such as input validation errors and insecure use of APIs. This analysis is considered "static" because it is performed without executing the code [20]. The group utilized GitLab SAST in the pipeline, employing the provided Go supported analyzer to examine the code for potential vulnerabilities. This continuous security feedback during the development process enabled the group to address security issues early on, reducing the risk of security-related problems in the final product.

**GitLab Secret Detection**
GitLab Secret Detection scans the repository for potentially sensitive information, such as passwords, API tokens, and other credentials. This feature helps detect if secrets are committed and pushed to the repository. When a secret is detected, the secret should be revoked and changed. This can help reduce the risk of unauthorized access or data breaches that might occur if secrets are stored in the code repositories [19].

**GitLab Dependency Scanning**
GitLab Dependency Scanning analyses the dependencies for known vulnerabilities. It checks the third-party libraries, packages, and modules used in the project against a database of known vulnerabilities, alerting if any of the dependencies have reported security issues [18]. This feature helps to stay up-to-date with the security status of the

dependencies in the project and allows the group to take appropriate action, such as updating to a more secure version or finding an alternative solution. This continuous scanning approach ensures that the project remains secure as new vulnerabilities are discovered and the list of dependencies is updated.

**Build**
In the GitLab pipeline, a build stage was implemented. This ran exclusively on the master branch or when a commit message contained ”Build executable”. It defines a job that compiles the Go code and creates a binary named orchestrator. The build stage uses the following build command:
`go build -a -ldflags ’-extldflags "-static"’ -o orchestrator`.

The flags[22] in the command provide the following:

- `-a`: Forces the rebuilding of all packages
- `-ldflags ’-extldflags "-static"’`: Links the binary statically
- `-o orchestrator`: Specifies the output file name as orchestrator

Incorporating this build stage into the pipeline automated the build process to ensure a consistent build environment. In addition, it simplifies artifact management by storing the built binary as an accessible artifact for later pipeline stages. Furthermore, it promotes a standardised and automated build process.

### 6.4.2 Ansible Pipeline

ansible-lint is integrated into the GitLab pipeline in the Ansible repositories and provides continuous style and quality checks. It is a community-driven CLI linting tool that checks Ansible playbooks, roles, and collections for patterns and behaviours, and that it follows best practice [45]. The tool ensures that the code adheres to the community standard and can help identify bugs. For instance, if a name is written with an incorrect convention, such as a lower case when it should be upper case, the lint tool will trigger an `name[casing]` message. ansible-lint enhances the quality of Ansible configurations and also makes the code easier to maintain.

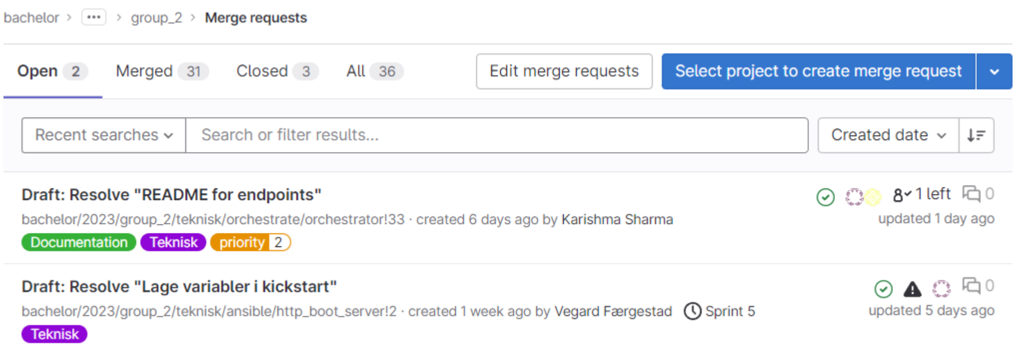
### 6.4.3 Employer Pipeline

Refer to Appendix I for information about the pipeline integrated by the employer and a visualization of the results.

## 6.5 Merge Requests

The group linked issues to merge requests in GitLab by creating a draft merge request with a dedicated branch in the project repository (see Figure 24). This method enabled group members to stay informed about merge requests in progress and promoted a clear separation of these throughout the development process. By linking branches to issues, the group maintained a structured workflow and minimized merge conflicts, as each branch focused on distinct tasks. The merge request inherited the labels from the linked issue, effectively highlighting the priority of the merge. This approach ensured smooth collaboration and efficient progress during the development process.

[22] The specifications of the build command can be found here: https://pkg.go.dev/cmd/go#hdr-Compile_packages_and_dependencies, visited 20.05.2023.



bachelor › ••• › group_2 › **Merge requests**

Open 2 | Merged 31 | Closed 3 | All 36 | Edit merge requests | Select project to create merge request

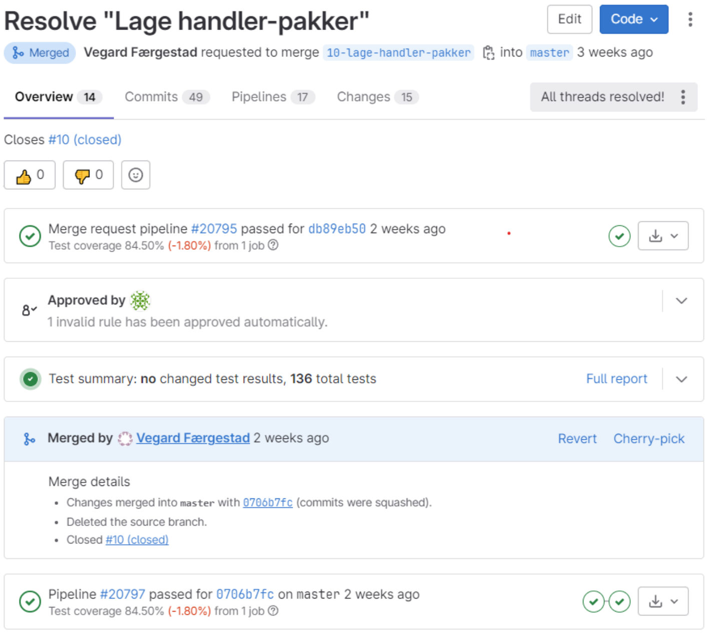
Recent searches | Search or filter results... | Created date

**Draft: Resolve "README for endpoints"**
bachelor/2023/group_2/teknisk/orchestrate/orchestrator!33 · created 6 days ago by Karishma Sharma
Documentation | Teknisk | priority 2
1 left 0
updated 1 day ago

**Draft: Resolve "Lage variabler i kickstart"**
bachelor/2023/group_2/teknisk/ansible/http_boot_server!2 · created 1 week ago by Vegard Færgestad Sprint 5
Teknisk
0
updated 5 days ago

Figure 24: Example of merge request drafts.

The merge requests required approval from a group member who did not contribute with any commits to the branch being merged (see Figure 25). If changes were committed after approval but before merging, the merge request needed to be approved again. This ensured unbiased code review and maintained code quality and readability. In addition, the group used GitLab pipelines when pushing to branches, ensuring all tests passed before merging (see Section 6.4). Changes were never pushed to the master branch without a merge request. The group encouraged squashing commits during merges to the master branch, to have a more organized commit history. When a merge request was approved and successfully merged, the linked issue was automatically closed. This ensured an organized and efficient way of managing completed issues and tracking progress. These practices guaranteed a quality check by at least one group member prior to merging and maintaining a clean repository.



**Resolve "Lage handler-pakker"**
Edit | Code

Merged **Vegard Færgestad** requested to merge `10-lage-handler-pakker` into `master` 3 weeks ago

**Overview 14** | Commits 49 | Pipelines 17 | Changes 15 | All threads resolved!

Closes #10 (closed)

👍 0 | 👎 0

Merge request pipeline #20795 passed for `db89eb50` 2 weeks ago
Test coverage 84.50% (-1.80%) from 1 job

**Approved by**
1 invalid rule has been approved automatically.

Test summary: **no** changed test results, **136** total tests | Full report

**Merged by** Vegard Færgestad 2 weeks ago | Revert | Cherry-pick

Merge details
- Changes merged into `master` with `0706b7fc` (commits were squashed).
- Deleted the source branch.
- Closed #10 (closed)

Pipeline #20797 passed for `0706b7fc` on `master` 2 weeks ago
Test coverage 84.50% (-1.80%) from 1 job

Figure 25: Example of an approved and merged merge request.

## 6.6 Other Quality Measures

This section will discuss additional measures undertaken to guarantee that the system operates at the desired quality level. These measures include PCAP analysis and adherence to the CIS NGINX benchmark.

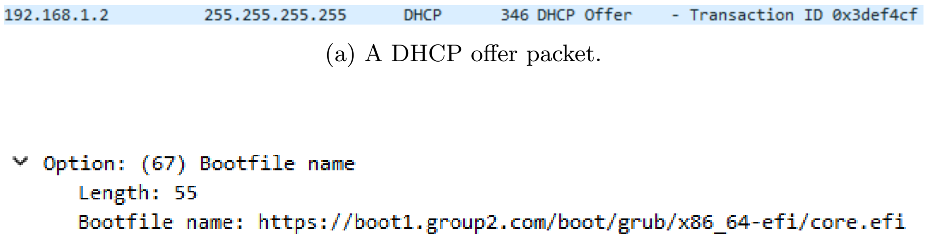
### 6.6.1 PCAP Analysis

To verify that network traffic was transmitted as intended, a packet capture (PCAP) was conducted from the perspective of the network booting server, which hosted the web server, the orchestration service, DNS, and DHCP services. By analyzing the data obtained from the PCAP file, the various hosts were identified, as specified in Table 5. This analysis provided valuable insights into the network traffic and confirmed that communications were functioning as designed. The PCAP was analyzed with wireshark[23].

| Host | IP address |
|---|---|
| Provision-server | 192.168.1.2 |
| Client | 192.168.1.3 |

Table 5: The IP addresses of the hosts in the packet capture.

The first part of the communication during the provisioning process is the DHCP traffic where the client retrieves its network configuration and NBP location from the boot server. The client starts by sending a broadcast DHCP Discover message to identify DHCP servers. The server responds with the DHCP offer, which includes the IP address, network mask, gateway, and the NBP location through DHCP option 67 (see Figure 26). The DHCP handshake is finished with the DHCP Request and the DHCP Acknowledge packets. As a result, the client has network access and is able to boot through the network boot server.



```
192.168.1.2          255.255.255.255      DHCP      346 DHCP Offer    - Transaction ID 0x3def4cf
```

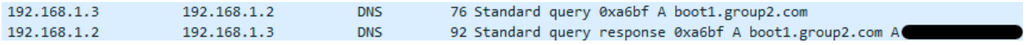
(a) A DHCP offer packet.

```
⌄ Option: (67) Bootfile name
    Length: 55
    Bootfile name: https://boot1.group2.com/boot/grub/x86_64-efi/core.efi
```

(b) The boot file option in the DHCP offer.

Figure 26: The DHCP offer packet with the boot file option.

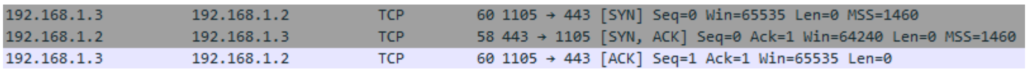
Since the NBP location is delivered through a Fully Qualified Domain Name (FQDN), the next step for the client is to resolve the corresponding IP address. This is done through a DNS query to the DNS server running on the network server. This traffic can be seen in Figure 27.



```
192.168.1.3          192.168.1.2          DNS        76 Standard query 0xa6bf A boot1.group2.com
192.168.1.2          192.168.1.3          DNS        92 Standard query response 0xa6bf A boot1.group2.com A [redacted]
```

Figure 27: The client's DNS query and the server's response. The answer is redacted in this in this figure.

The client will then try to initiate a HTTPS session with the web server, using the IP address it got from the DNS server. The first part of a HTTPS session, is the establishment of a TCP session, through a TCP handshake, which can be seen in Figure 28.

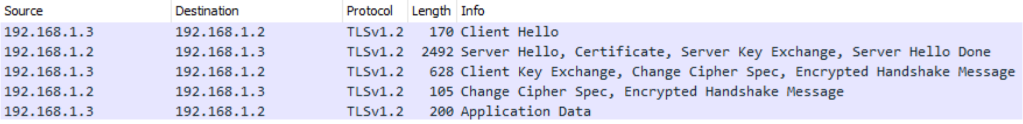


```
192.168.1.3          192.168.1.2          TCP        60 1105 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
192.168.1.2          192.168.1.3          TCP        58 443 → 1105 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
192.168.1.3          192.168.1.2          TCP        60 1105 → 443 [ACK] Seq=1 Ack=1 Win=65535 Len=0
```

Figure 28: The TCP handshake between the client and the web server.

[23] https://www.wireshark.org, visited 16.05.2023.

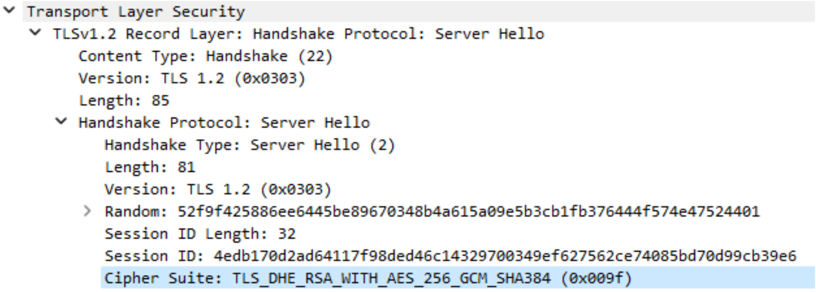
It is crucial to ensure that confidential traffic is encrypted. All communication with the web server is conducted using HTTPS, which employs the TLS protocol to facilitate encryption. A TLS session is characterized by the initial TLS handshake, followed by encrypted packets containing the actual data. The individual TLS handshakes between the various hosts were identified in the PCAP, ensuring that secure and encrypted communication was maintained throughout the system's operation.



```
Source           Destination      Protocol  Length  Info
192.168.1.3      192.168.1.2      TLSv1.2    170 Client Hello
192.168.1.2      192.168.1.3      TLSv1.2   2492 Server Hello, Certificate, Server Key Exchange, Server Hello Done
192.168.1.3      192.168.1.2      TLSv1.2    628 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
192.168.1.2      192.168.1.3      TLSv1.2    105 Change Cipher Spec, Encrypted Handshake Message
192.168.1.3      192.168.1.2      TLSv1.2    200 Application Data
```

Figure 29: The TLS handshake between the client and the web server.

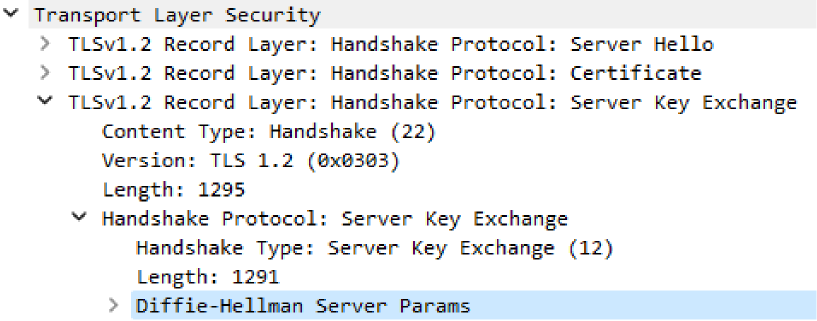
A TLS handshake between the client and the server can be observed in Figure 29. The handshake illustrates that the client initiates the session when it receives the boot file location from the DHCP server and attempts to retrieve both the OS ISO-image and the Kickstart file. This process occurs on multiple occasions, as the client's registration and state updates to the orchestration service are executed in the same manner.



```
⌄ Transport Layer Security
  ⌄ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 85
    ⌄ Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 81
        Version: TLS 1.2 (0x0303)
      › Random: 52f9f425886ee6445be89670348b4a615a09e5b3cb1fb376444f574e47524401
        Session ID Length: 32
        Session ID: 4edb170d2ad64117f98ded46c14329700349ef627562ce74085bd70d99cb39e6
        Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
```

Figure 30: A TLS server hello specifying the Diffie-Hellman cipher.

As mentioned in Section 3.2.2, the web server is configured to exclusively use the most secure ciphers that are supported by the client. One reason for this configuration is to ensure forward secrecy, which guarantees that encrypted data remains secure and cannot be decrypted even if the server's private key is compromised in the future. A widely adopted cipher that provides this assurance uses Diffie-Hellman as its key exchange algorithm[24]. This cipher was employed during these handshakes, as depicted in Figure 30 and 31.



```
⌄ Transport Layer Security
  › TLSv1.2 Record Layer: Handshake Protocol: Server Hello
  › TLSv1.2 Record Layer: Handshake Protocol: Certificate
  ⌄ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 1295
    ⌄ Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 1291
      › Diffie-Hellman Server Params
```

Figure 31: A TLS Server Key Exchange packet specifying the Diffie-Hellman parameters.

[24] https://www.ibm.com/docs/en/zvse/6.2?topic=SSB27H_6.2.0/fa2ti_openssl_consider_diffie_hellman.htm, visited 14.05.2023.

### 6.6.2 CIS Benchmark

To ensure the security and quality of the NGINX deployment, the NGINX Center for Internet Security (CIS) benchmark was conducted. The benchmark, or hardening guide, consists of recommendations from cyber security experts to help ensure an acceptable baseline security level for NGINX [34]. Some of the recommendations included securing access control, implementing proper logging, and enabling security-related HTTP headers. For this project, version 2.0.0 of the NGINX benchmark was used[25]. The benchmark consists of multiple levels with different requirements, based on how secure the deployment aims to be. Level 2 hardening is an extension of the level 1 hardening to improve security. However, the level 2 hardening might cause performance or compatibility issues. In addition to the levels, there are different profiles based on the different functionality of NGINX. There are different recommendations based on if NGINX is set up as a reverse proxy or a web server. For this project, the aim was to fulfil the benchmark level `Level 2 - Webserver`, however, all recommendations were not followed (see Section 7.4.1).

To ensure that the web server had the proper implementations as described in the benchmark, the steps in the audit section of the report were followed. This section described either manual or automatic steps to verify that the system complies with the recommendation. Additionally, some of these checks and their remediation were implemented into the Ansible role so that its compliance was automatically verified and fixed, if necessary. Refer to Appendix S for details regarding which recommendations were followed.

## 6.7 Evaluations by the Employer

Various evaluations of the solution were conducted, both during the development phase and of the final product. These evaluations were conducted by professionals connected to the employer's IT department and included discussions about architecture and implementation, evaluation of the security assessment, and user tests of the final product. These evaluations were invaluable and provided the group with insight into the quality of the project.

### 6.7.1 Architecture and Implementation

In the final stages of the implementation of the project, the solution was presented to architects and other stakeholders at the employer. This step aimed to inform the stakeholders of the new system that was to be deployed into production at a later stage. In addition, this step was taken to ensure that the project did not include any major flaws or difficulties that would make it harder for the employer to accept and deploy the provisioning system into their environment.

Overall, most architects were satisfied with the solution. They especially appreciated the use of UEFI HTTPS boot for increased security. However, it was noted that an automatic system like this might introduce a lot of maintenance. Since the process is defined in scripts, many of these might have to be updated regularly to reflect the desired functionality of the client. In addition, a comment was made that an automated provisioning system might make it more time-consuming to troubleshoot a failed provisioning. Therefore, it is important to have a strict quality assurance process for the configurations used in the provisioning system.

There were also comments regarding the automatic BIOS configuration. Some mentioned that this automatic configuration might introduce some vulnerabilities and increase the attack surface of the clients. In addition, Intel's Out-Of-Band management tool[26] was

[25] https://www.cisecurity.org/benchmark/nginx, visited 13.05.2023.

[26] Only works on Intel vPro CPUs. https://www.intel.com/content/www/us/en/business/enterprise-computers/resources/out-of-band-management.html visited 12.05.2023.

suggested as a way of handling the automatic BIOS configuration.

The group's initial plan was to use Caddy instead of NGINX for the web server at the time of the presentation, some concerns were raised regarding the use of Caddy since it was not part of the standard library of RHEL 9. It was also an unknown web server for most of the architects and they encouraged the use of NGINX.

Another suggestion from the presentation was to make the system easy to use and the process well documented. This would in turn enable the help desk, people with limited access to the environment or people with limited knowledge regarding provisioning Linux clients to provision new clients, using the provisioning system with low effort.

One gripe they had with the system, was the use of the orchestration service, since this component adds custom code that needs to be maintained, in addition to increasing the complexity of the provisioning system. They suggested that the group should consider keeping the automatic installation process as one stage as is and that an operator manually starts the process of executing the Ansible scripts.

In conclusion, the presentation to the architects and stakeholders provided valuable feedback for the project's development. Despite the concerns raised, they were satisfied with the solution. Addressing the maintenance and usability concerns, as well as the implementation of the suggested changes, enhanced the quality of the provisioning system.

### 6.7.2 Security Assessment

The security assessment was evaluated by a Security Architect (SARC) upon completion. This included an evaluation of the BIA, threat assessment and risk assessment. In general, the feedback was overall positive and mentioned that the framework was sufficiently followed. The BIA, with its justifications, was described as well assessed and documented, reflecting the purpose, objective and rationale of the values set for the provisioning system. Moreover, the threat assessment was evaluated as thorough with the use of MITRE ATTACK[27], and with good systematic visualisation. Lastly, even though the SARC did not have detailed knowledge about the final solution, it was given an overall evaluation of the risk assessment. The evaluation included that the assessment covered the relevant risk scenarios tied to the system and the associated vulnerabilities.

Furthermore, the evaluation indicated areas that could be improved. In particular, certain aspects could have been made more explicit, and some formulations could have been worded differently. Nonetheless, the employer evaluated and approved the security assessment conducted. Further discussion on this is found in Section 7.5.1. The full evaluation can be found in Appendix G.

### 6.7.3 User Testing

To get a professional assessment of the solution and its related documentation, user tests were conducted by employees from the IT department at the employer. These tests aimed to evaluate the effectiveness of the automatic provisioning system and compare its security, flexibility, workload and time efficiency with the current manual provisioning process, as well as provide feedback on the related documentation. Considering the time aspect, the user test and the current process are not completely comparable, as the Ansible playbook used during testing differs from the one the employer will use in production. However, it still provides a sense of the system's usability, and the process's time efficiency, especially for the parts excluding the playbooks. Furthermore, the number of relevant/suitable employees was limited, and therefore the user test focused on qualitative insights rather than quantitative (see Section 7.6 for further elaboration).

The scope of the test was the whole provisioning process, including the GRUB NBP,

[27] https://attack.mitre.org/, visited 16.05.2023

AAP, and the orchestration service. Out of scope was the UEFI configuration, and the deployment of the provisioning system. The tests assumed that one had a general knowledge of the relevant background, including the current manual process and provisioning in general. This was a requirement that was necessary for a good professional evaluation. The user tests gathered feedback on the entire process, logs, and other aspects. In regards to the user stories established, all were executed in the tests, except for the user story related to provisioning multiple clients simultaneously, as the employer only had one client available for testing purposes.

The questionnaire consisted of two parts. The first part was a closed question where the answer should be a score between 1-10, where **1** is *Very Easy* and **10** is *Very Difficult*. The last part consisted of open questions for more elaborate answers. The user test was conducted on three different participants. Testing with a larger number of participants would have been ideal, and it is important to keep this in mind when evaluating the results. Regardless, the feedback received offers a good indication of the quality of the solution. Refer to Appendix J for the complete user tests.

The first participant, a senior security consultant, reported a significant reduction in time taken by the new system compared to the manual process and found the documentation easy to follow. Furthermore, the feedback stated that the solution was a huge improvement in regards to security, and much easier to maintain and further develop. The difficulty level of each tested component was rated a score of 1 - *Very Easy* to perform.

The second participant, a senior security engineer, found the automatic provisioning system's installation process easy (rated 2 out of 10) but suggested improvements in the documentation. They didn't interact with the API and configuration addition, hence no feedback was provided for these areas. Logs were rated as easy to understand (rated 2 out of 10), with a suggestion for hex-encoded logs for security. The new system was described as faster, more secure, user-friendly, and easier to maintain compared to the current one.

The third participant, a junior security consultant, found the installation process and using the API straightforward. They suggested more specific documentation and adding progress tracking to the API. They found the logs easy to understand but suggested adding colour coding and improving terminology consistency across the system. Adding a new configuration was moderately easy, with a few suggestions for better documentation.

Overall, the automatic provisioning system was found to be a significant improvement over the current manual process. The participants noted increased efficiency, improved usability, and a sufficient logging system. In general, all aspects of the solution were easy to use. Nonetheless, they also gave feedback for improving the documentation and other aspects. This will be further elaborated in Section 7.6. In summary, the user testing provided a professional evaluation of the automatic provisioning system. It not only identified areas for future improvement but also confirmed the effectiveness. Furthermore, proved to meet expectations from the group in terms of resource usage, user-friendliness and security, which also is a great improvement from the existing manual process.

# 7 Results and Discussion

This section discusses the project's technical results, referencing the task specifications outlined in Section 2.1. In addition, it includes a discussion on the chosen architecture and implementation, potential alternative solutions, and the rationale behind the decisions. Further, the security assessment and user tests will be discussed. The section concludes with reflections on our experiences throughout the project process.

## 7.1 Technical Results

A set of specifications were set at the beginning of the project (see Section 2.1). These requirements laid the foundation for the development process of the provisioning system. The results indicate that the majority of the specifications were realised through this project. This section will discuss which specifications were realised, and which were just partially, or not, realised. The specifications were split into functional, non-functional and operational specifications. Each section will consist of tables summarising the realisation of the specifications, together with further discussion.

### 7.1.1 Functional

The functional specifications were prioritised into two categories, priority one for specifications that had to be present and priority two for non-vital specifications.

**Priority One**
For a summary of the priority one realisation, see Table 6.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| Secure network booting of physical clients | | X | |
| Automatic installation of Red Hat Enterprise Linux (RHEL) 9 | X | | |
| Automatic execution of Ansible playbooks to configure the client | X | | |
| Automatic verification of the Operating System install | X | | |
| Non-critical errors should be logged, whilst critical errors should stop the provisioning process. | X | | |

Table 6: Showing the priority one functional specifications, and to what degree they have been realised.

The implementation of UEFI HTTPS boot aims to address the specification of *Secure network booting of physical clients*. The system is primarily implemented with compatibility with Dell clients in mind, but this does not necessarily exclude compatibility with other vendors' clients. When using HTTPS as the boot method, all traffic is encrypted using TLS. However, since the provisioning system is deployed in an air-gapped network, some of the integrity aspects of the TLS certificates cannot automatically be verified, since Dell's implementation of UEFI does not automatically trust any certificates. As a result, the user must provide a certificate to the client manually. This improves the security and integrity of the network booting. However, it reduces the usability of the system since the user still most likely needs a USB drive to deploy the certificate. Moreover, enabling secure boot would also help improve the security of the network booting, since secure boot ensures that the boot program is signed by a trusted source. This aims to ensure the integrity of the code running during the start-up phase [11].

One stage of the network booting involves booting into the network version of the GRUB boot loader (NBP). This helps increase the usability of the system, as it provides multiple configuration options to the user, but it also increases the complexity and attack surface of the system. To somewhat minimise this, unneeded modules were removed, including the non-UEFI version of the NBP. To secure GRUB even further, authentication can be added. This ensures that the user needs to enter a password before being presented with the GRUB menu. However, this was not implemented in this iteration of the provisioning system.

As a result of these additional security measures that could have been implemented, this specification is partially realised.

The system provides an interface for automatically installing RHEL through the use of Kickstart files. This aims to address the specification of *Automatic installation of Red Hat Enterprise Linux (RHEL) 9.* These files specify the answers to the questions normally prompted during the installation process and provide them to the installer. Kickstart files are the standard way of providing automatic installations of RHEL 9 [25]. The Kickstart file is provided through the web server and automatically provided to the kernel through the `inst.ks` kernel parameter when booted from the GRUB NBP. Therefore, the specification is realised.

To automatically configure the clients using Ansible playbooks, a deployment of Ansible Automation Platform (AAP) has been integrated into the system. This aims to fulfil the specification of *Automatic execution of Ansible playbooks to configure the client.* It would also be possible to push the configuration from the orchestration service or execute the scripts locally on the client. These solutions however would make it harder to handle secrets properly and make the process more error-prone. Since Red Hat already has an application that solves these issues, AAP, it made sense to integrate the provisioning system with it instead. Another advantage of using AAP is that it ensures the integrity of the scripts run. In addition, Ansible playbooks are used to meet the specification of *Automatic verification of the Operating System install*, by providing a consistent way to configure and check the clients. It also means that less code is needed in the orchestration service since it only needs to handle the execution of one type of script. These two specifications are realised.

To fulfil the specification *Non-critical errors should be logged, whilst critical errors should stop the provisioning process* most stages of the system are designed to log events, including errors. In the event of a critical error, the orchestration service captures and logs the error. Additionally, the error is added to the status endpoint, and the provisioning process is halted. However, not all phases of the provisioning process are able to signal the orchestration service and stop the provisioning process. This might make the provisioning process a bit harder to follow since the user must recognise that the process is not continuing as expected. Despite this, the specification is considered realised.

### Priority Two

For a summary of the priority two realisations, see Table 7.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| The automated installation should be backwards compatible with version 8 of RHEL | X | | |
| The system should support reprovisioning clients | X | | |
| Automatic configuration of the BIOS | | | X |

Table 7: Showing the priority two functional specifications, and to what degree they have been realised.

The specification *The automated installation should be backwards compatible with version 8 of RHEL* is addressed by the user being able to provide the OS ISO-image, Kickstart file and Ansible playbooks for RHEL 8, and use the system for automatic provisioning. This is the case for most Linux distributions that supports Kickstart files. The system can also be modified to support other types of Operating Systems, such as Debian[28] with the use of a preseed file[29]. To accomplish this, a small modification in the GRUB menu Ansible tasks must be done. In its current form, some Red Hat family distribution-specific kernel parameters are automatically provided when generating the GRUB menu using Ansible. Therefore, the specification is realised.

Regarding *The system should support reprovisioning clients* specification, the system handles a reprovisioning of a client in the same manner as a regular provisioning process. No extra security measures are currently implemented, since by default the storage volumes on the clients are usually encrypted in the deployment environment of an organisation. As long as these keys are changed and the previous key is destroyed, there should be a very low risk of being able to extract data from the previous installation. In addition, the existing TRIM function[30] on the SSD ensures that the deleted data is removed from the physical data blocks on the SSD. Consequently, the specification is realised through a combination of the system's functionality, and existing Operating System functionality.

The last functional specification, *Automatic configuration of the BIOS*, specifies that the BIOS should be configured automatically. Implementing this would make the entire provisioning process automatic. Furthermore, it would simplify the process of executing the UEFI HTTP boot process by automatically applying the correct settings and certificate needed for the provisioning. Multiple solutions are available to solve this task, such as Intel's Out-Of-Band management tool[31] and Dell Client Configuration Utility[32]. However, the primary issue with these solutions is that they currently only support Microsoft Windows. Since the deployment environment consists of RHEL devices, these first-party solutions were not a possibility. Therefore, it was decided that this feature was not to be implemented since the group was unable to find a solution that supported RHEL. In addition, a solution like this might introduce a lot of maintenance, since various client vendors and models might have slight differences which need to be taken into account when automating the tasks. This concern was also raised through the evaluation by the employer (see Section 6.7.1). As a result, this specification is not realised.

### 7.1.2 Non-functional

The non-functional specifications are divided into deployment and security and privacy specifications, as well as usability and maintainability, reliability, and performance specifications.

**Deployment**
For a summary of realisation, see Table 8.

[28] https://www.debian.org, visited 21.05.2023.

[29] https://wiki.debian.org/DebianInstaller/Preseed, visited 21.05.2023.

[30] https://www.crucial.com/articles/about-ssd/what-is-trim, visited 21.05.2023.

[31] Only works on Intel vPro CPUs. https://www.intel.com/content/www/us/en/business/enterprise-computers/resources/out-of-band-management.html, visited 12.05.2023.

[32] https://www.dell.com/support/kbdoc/no-no/000179457/what-is-client-configuration-toolkit?lang=en, visited 12.05.2023.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| The preferred deployment type is on one or multiple virtual machines (VMs) | X | | |
| The system should be deployable on RHEL version 9.x | X | | |
| The deployed system should function without an internet connection | X | | |

Table 8: Showing the non-functional deployment specifications, and to what degree they have been realised.

Two deployment specifications state that *The preferred deployment type is on one or multiple virtual machines (VMs)* and *The system should be deployable on RHEL version 9.x.* Using the Ansible playbook `http_boot_server`, a system administrator is able to choose which provisioning component to deploy on which server using tags in the inventory file. Moreover, the system can be deployed on bare-metal servers or using containers as well, but these methods are not directly supported. In addition, the deployment of the system is tested and verified to work on RHEL 9, but it should work on most Red Hat family Linux distributions that come with SELinux. Therefore, the specification is realised.

In regards to the last specification, *The deployed system should function without an internet connection*, the system supports not having an internet connection. However, this requires that all desired packages are provided in the OS ISO-image and that the Ansible scripts do not have any external dependencies for the provisioning process to work as intended. The system itself does not need an internet connection to provision the clients. Hence, the specification is realised.

**Security and Privacy**
For a summary of realisation, see Table 9.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| Code should be analysed by the employer's internal code pipeline | X | | |
| The system should be security assessed | X | | |
| The image of the operating system (OS ISO-image) should be fetched from an approved secure location | | X | |
| Network traffic that can be encrypted, should be encrypted | X | | |
| A Public Key Infrastructure (PKI) should be used | X | | |
| Access to the system should be sufficiently secured | X | | |
| The integrity of the scripts used to configure the clients should be ensured | X | | |
| The system should use secure default values | X | | |

Table 9: Showing the non-functional security and privacy specifications, and to what degree they have been realised.

*Code should be analysed by the employer's internal code pipeline* aims to ensure security and quality (see Section 6.4.3). The code of the orchestration services written in Go has been analysed, However, the Ansible code has not been analysed by the pipeline, since the tools integrated into the pipeline do not support the Ansible language. Therefore the linting tool ansible-lint was used instead. This decision was made in collaboration with the employer. Thus, the specification is realised.

A security assessment has been conducted, as specified in the specification: *The system should be security assessed.* The assessment located in Section 5 primarily focused on the security associated with the system as a whole. Moreover, the tools used in the provisioning system have been indirectly assessed through this security assessment. In addition, security was in mind when choosing the tools, indirectly acting as a security assessment. The vulnerabilities introduced by DNS and DHCP were explicitly assessed since these features were not strictly needed for the provisioning system to function. In light of this, the specification is realised.

*The image of the Operating System (OS ISO-image) should be fetched from an approved secure location*, specifies that the system should fetch the OS ISO-image which is not the case in the current implementation of the system. The system administrator must manually move the OS ISO-image and the related files to the HTTP file server. This simplifies the configuration and reduces the attack surface of the system. A system that fetches the files from another file server could have been implemented, but it does not make the configuration any simpler. If the files have to be moved to a file server with a specific layout, it might as well be the one used by the provisioning system. It was also considered to use Ansible to deploy the files, but it was decided against because the files could not be stored in git because of file size limitations in centralized repositories[62]. In addition, moving and checking the files for each run with an Ansible script would probably be too time-consuming. As a result, this specification is partially realised.

*Network traffic that can be encrypted, should be encrypted* specification is realised. For an analysis of the network packets see Section 6.6.1. To secure the HTTP traffic, TLS certificates were used, which is an implementation of PKI, which in turn realises the second specification, *A Public Key Infrastructure (PKI) should be used.* DNS and DHCP are currently the only components of the system where the network traffic is not encrypted. DNS supports encryption, however in a limited fashion. It would be a possibility to use DNS-Over-HTTPS[33] to encrypt the DNS traffic, but there is some uncertainty regarding support by the UEFI clients trying to network boot. Attempts were not conducted during this project, but it might be a future feature to test out. DHCP, on the other hand, cannot be encrypted since it sends broadcast traffic to all available clients on the network.

The *Access to the system should be sufficiently secured* specification is primarily handled by physical access control to the network. The services of the provisioning system do not support an access control system. The web server component cannot have access control, since it would be impossible to boot into an image on the server [34]. Some parts of the orchestration service could probably have access control, but this would create a more complex Kickstart file. The GRUB NBP does have support for adding password authentication, which in turn would make it a bit harder to exploit the system. However, since all the images and configurations used by GRUB are available to the network, the effect of this feature is minimal. Since the network is only accessible from inside the employer's network, in addition to only being available on specific physical ports, it is currently deemed a sufficient access control to the system. Furthermore, the level of access control was assessed as sufficient in the security assessment (see Section 5). Given these conditions, the specification is realised.

Since the automatic provisioning system uses AAP for executing the Ansible playbooks on the clients, the integrity of these files are handled by AAP. As a result, the specification *The integrity of the scripts used to configure the clients should be ensured* is realised,

[33] https://www.rfc-editor.org/rfc/rfc8484, visited 21.05.2023.

[34] Specialized boot NBPs might support this, but it was not considered for the scope of this project.

while also making the provisioning system less complex, since an external party handles this specification.

In regards to the specification *The system should use secure default values*, default values are set to a secure value when possible. For instance, the NGINX Ansible role sets a default of the SSL versions to be `TLS1.2 TLS1.3` instead of a more insecure value. An example from the orchestration service is that if no logging information is specified, it will create a folder containing the logs in the working directory with the log level `info`. This ensures that it always is possible to view the logs of the orchestration service. However, there might be cases where the defaults not necessarily are secure or cases where what is deemed secure has changed. Therefore it is vital to occasionally verify that the defaults are still of the desired security level. Nevertheless, the specification is considered realised.

**Usability and Maintainability**
For a summary of the realisation, see Table 10.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| The configuration of the system should be defined in code using Ansible | X | | |
| The system should be scalable, preferably horizontally | X | | |
| The system should be documented sufficiently | X | | |
| The code should follow best practices and be easily maintainable | X | | |
| The system should make it easy to add new provisioning configurations | X | | |

Table 10: Showing the non-functional usability and maintainability specifications, and to what degree they have been realised.

The configuration of the automatic provisioning system is done using Ansible as stated in the specification *The configuration of the system should be defined in code using Ansible.* The whole system, with all components, is configurable from a single Ansible playbook. Some components of the system are defined in Ansible roles, used by the main playbook. Using Ansible roles in combination with a primary configuration playbook makes the system easier to configure, whilst still having small separate components that are easy to maintain. It also promotes the reuse of the different roles in other situations. However, some of the Ansible roles could be more flexible in terms of having more variables. This would enable the provisioning system to be deployable in more environments with different requirements. As a result, the specification is realised.

The specification *The system should be scalable, preferably horizontally* is addressed by all components being capable to be deployed on different servers, which helps enable the scalability of the system. Some of the components can have multiple instances running at the same time, making them scalable horizontally. However, some components will need a load balancer in front, to ensure that the traffic is routed to one of the multiple instances. Examples of this are NGINX and the orchestration service. However, the orchestration service would need the load balancer to send all requests from one provisioning job to the same instance every time. This is because the orchestration service does not currently have a way of sharing the state of a provisioning job between instances. For most deployments, there should be no need for multiple instances of the services, since they should be able to handle many requests simultaneously. In conclusion, the specification is realised.

To meet the *The system should be documented sufficiently* specification, the code for the orchestration service is documented with comments for each function and additional explanations where the code might be challenging to understand. This approach ensures that other developers can easily read and comprehend the code. Furthermore, the README.md file in the repository details how to use the REST API and configure the service. In regards to the deployment, the Ansible code serves as documentation. The declarative code specifies the expected server configuration, effectively acting as always up-to-date documentation. The README also outlines how to install the requirements, and what variables are used to configure the role/playbook, and includes an example playbook (for roles only). In conjunction with the more technical documentation, a user guide and a system administrator guide are provided. These guides offer step-by-step instructions for how to execute common tasks, aiming to prevent user errors, both during configuration and usage of the system. Hence, the specification is realised.

In regards to the specification *The code should follow best practices and be easily maintainable*, the code was developed to adhere to best practices. One example is to use the appropriate HTTP status codes in REST API. Additionally, some pipeline stages incorporate tools to ensure compliance with best practices. These tools include GitLab SAST and ansible-lint. Secrets are not directly managed within the project's code; instead, they are injected into it. For instance, an environment variable is used for the AAP instance. This approach helps avoid potential pitfalls associated with handling sensitive information. Thus, the specification is realised.

To ensure the maintainability of the system, the provisioning system is designed to simplify the process of adding new configurations. This addresses the specification *The system should make it easy to add new provisioning configurations.* For instance, adding a new boot configuration requires minimal steps. Documentation is provided to guide system administrators through the necessary steps, making it straightforward to implement new configurations. Moreover, the GRUB menu configuration is handled through Ansible variables, ensuring consistency with the rest of the provisioning system. However, adding files to the HTTP file server can be more involved, as it currently involves a manual process. The system administrator must move the desired files and unpack the OS ISO-image, as specified in the documentation. This step is likely to be more complex and error-prone compared to other aspects of adding a new configuration. Nonetheless, the specification is realised.

**Reliability**
For a summary of the realisation, see Table 11.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| The system should handle multiple provisionings at the same time | X | | |
| Every client should be provisioned consistently | X | | |

Table 11: Showing the non-functional reliability specifications, and to what degree they have been realised.

The provisioning system should support multiple provisionings at the same time, as stated in the specification *The system should handle multiple provisionings at the same time.* This is because the tools utilised are built to handle many connections at simultaneously since their primary use case is larger deployments. However, this claim is unverified since the group did not have multiple clients that support UEFI HTTP at the time of testing.

In spite of this, nothing indicates that this specification is not realised.

Choosing the same configuration in the GRUB boot menu should result in the same configuration for each install, as specified in the *Every client should be provisioned consistently* specification. This is ensured since the same code is being executed each time, therefore no significant differences should occur between installations. During the Kickstart process, no differences should occur, but when executing the Ansible playbooks some might fail and the process might still continue. This is because there is an option that makes it possible to continue the provisioning process even if an AAP job fails. This is implemented so that non-vital playbooks that might fail can be rerun at a later stage. If this happens the error will be logged and the client will be inconsistent with other clients. Regardless, the specification is realised.

**Performance**
For the summary of realisation, see Table 12.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| The provisioning should take less time than the current manual duration | X | | |

Table 12: Showing the non-functional performance specifications, and to what degree they have been realised.

The specification states that *The provisioning should take less time than the current manual duration.* The amount of time saved has not been calculated but it is significantly faster. This has been verified by various employees at the employer in Section 6.7. They suggested that the automatic provisioning system was significantly faster than the current process, and the specification is therefore realised.

### 7.1.3 Operational

For the summary of operational specifications realisation, see Table 13.

| Specification | Realised | Partially Realised | Unrealised |
|---|---|---|---|
| The system should create audit logs | | X | |
| The OS ISO-image file name and file hash should be logged upon provisioning | | X | |

Table 13: Showing the operational specifications, and to what degree they have been realised.

When it comes to the operational specifications *The OS ISO-image file name and file hash should be logged upon provisioning*, the group was only partially able to implement the desired specifications. Each component of the system creates audit logs containing what and when, but not necessarily who. Since there is no authentication mechanism, most of the services cannot specify which has started a provisioning process. It is however possible to identify which client has been provisioned through the MAC address logged in DHCP log and the IP address in the DNS, and NGINX logs. The second operational specification states that the OS ISO-image file name and file hash should be logged upon the provisioning process. This is currently not implemented because the provisioning system works with the unpacked ISO-image instead of the single file. The hashes of these files are not logged as part of this system either, since it is the RHEL installer environment that fetches these files. The different file names are however logged as part of the NGINX logs.

## 7.2 Architecture

The selection of provisioning system components, their development approach, and their dependencies constituted an early and crucial decision-making process. These choices carried implications for security, the development process, user-friendliness, and the overall quality of the final product. Consequently, extensive research was conducted beforehand to ascertain the possible alternatives and to make informed decisions that would optimise the provisioning system's performance and functionality.

When a solution needs to include multiple different components, the boundaries between the components must be chosen in terms of responsibility and deployment. On one hand, the solution can be set up as a "monolithic" stack, where all the responsibilities and functions live within a single service. This service is created, deployed, updated, and taken down as a single unit. There are some advantages to such a solution. Firstly, a monolithic stack is often easier to manage. All functions are in one place, within a limited space. Secondly, communication between the functions in the service is simple since they reside on the same unit and thus in the same memory. For example, the network boot server could easily start the configuration process in the Ansible Automation Platform (AAP), which in turn could retrieve any configurations directly from the web server's memory.

The problems with such a solution arise when it needs to be changed and updated in the future. When all functions are combined like this, every single change becomes difficult and riskful, as it can affect many other parts of the solution. This is called a large "blast radius". Adaptations also become challenging; when the client needs to familiarize themselves with the solution and adapt it to their environment, it may require significant resources to determine what changes need to be made and actually implement them. Such a solution often leads to a drastic decrease in the frequency of updates and changes [49], which in turn can compromise security.

Consequently, the architecture has been designed to establish clear boundaries between the various services, with an emphasis on the distribution of responsibilities. Each distinct service, including the web server, DNS server, DHCP server, and orchestration service, has been developed as an independent project. This approach ensures that as long as communication channels between the services are preserved, any individual service can be modified, updated, or replaced with minimal impact on the remaining services. Each component resides within its own GitLab project, where its development occurs.

This architecture also allows for the separate deployment of each service. However, this may not always be the desirable approach. In certain instances, it may be more practical and resource-efficient to deploy a larger portion of the infrastructure simultaneously. Given that the service is deployed using Ansible, each component is defined in its own Ansible Role (see Figure 11). This arrangement accommodates both the need for joint deployment and independent development; striking an optimal balance between flexibility and integration.

As previously discussed, establishing communication between components in such a solution becomes more complex. This complexity necessitated the development of a method that enables components to exchange essential messages, including new client registrations, state updates, and initiating the subsequent configuration step. Allowing direct communication between each component was impractical, as it did not provide a centralized location for logging all interactions. Consequently, a separate component was required to facilitate communication and ensure that all exchanges were logged in an easily accessible manner. Multiple of the components this service needs to connect and communicate with, use secrets to ensure authorization. Since the communication is orchestrated through a single service, all the different secrets can be stored in this single service, instead of being distributed to multiple locations.

Multiple of these responsibilities could have been done manually, as suggested in the evaluation by the employer (see Section 6.7.1), especially the initialising of the next stage when needed. However, this would have made the solution require significantly more manual interaction, which in turn would affect the realisation of the functional specifications (see Section 7.1.1). This would also make it more difficult to achieve a satisfying level of logging, as each component had to keep logs of the events on its own. Considering these factors, there were decided that there were no suitable existing alternatives to address this need, primarily due to the task's specificity to the requirements of the various components within the solution. As a result, it became necessary to develop custom software with the requisite functionality.

Developing custom software introduces a range of new challenges. As mentioned earlier, it was desirable to create components that could be easily maintained in the future. Utilizing third-party software often includes the benefit of ongoing maintenance provided by the developers. The challenge with custom solutions, however, is the need to allocate resources for actively updating and improving the software to ensure functionality and security. This demands considerable effort from both the initial developers (in terms of readable code and documentation) and those responsible for subsequent maintenance. In this case, the developers and maintainers are not the same individuals, further increasing the issue. Nevertheless, in this particular scenario, the custom solution was deemed the most appropriate.

When provisioning a client, boot options must be delivered through an appropriate method. One approach is to manually enter parameters in the BIOS during the client setup. However, this method is prone to errors, as users must remember and accurately input the correct settings. Moreover, it does not fulfil the requirement of automation. A more efficient alternative is to utilize an external component to automatically deliver a Network Boot Program (NBP) location to the client, with the most common method being through DHCP. When a new client connects to the network managed by the DHCP server, the server can transmit the NBP location along with network information. This approach minimizes the potential for human error and offers a more automated process, thereby enhancing the overall efficiency and reliability of the provisioning procedure (see Table 11).

A notable drawback of DHCP is that it relies on ”broadcast” traffic for communication. Consequently, all packets transmitted are sent to every device on the network, not just the intended recipient. This exposes the NBP location to all connected devices, thereby creating a new attack vector and expanding the system's attack surface. Despite this vulnerability, DHCP was deemed necessary for the system's functionality, and its associated risks were assessed in conjunction with the overall system in Section 5. This evaluation ensured that the potential security implications were carefully considered and addressed to maintain an acceptable level of risk.

Whether to utilize a DNS server as a component was also a topic of discussion. Having a DNS server in the network simplifies specifying hosts' locations since a human-readable domain name can be used instead of the host's IP address. This, however, increases the system's attack surface and introduces several potential drawbacks.

Firstly, implementing a DNS server adds complexity to the system, which may lead to increased maintenance and troubleshooting efforts. Secondly, it exposes the system to potential security threats, such as DNS cache poisoning and Distributed Denial of Service (DDoS) attacks. These attacks can disrupt the system's availability, causing downtime and potential data loss. Additionally, the reliance on a DNS server can introduce a single point of failure, making the entire system vulnerable if the server becomes compromised or experiences technical issues.

Despite these potential negatives, the decision was made to incorporate a DNS server into the system. The benefits of using human-readable domain names for specifying host locations and the ease of managing host addresses outweighed the potential drawbacks. Furthermore, by implementing proper security measures and redundancy strategies, the risks associated with the DNS server can be mitigated, ensuring the overall stability and security of the system.

The outcome of these discussions resulted in the solution's architecture, which is thoroughly discussed in Section 3.1.

## 7.3 Choice of Tools

This section will discuss which tools that could have been used for each component, including their advantages and disadvantages. In addition, the section will justify the choices of tools that were used in the solution.

### 7.3.1 Web Server

| Tool | Advantages | Disadvantages |
|---|---|---|
| httpd | Support by Red Hat [24]<br>Widely used | Many unneeded features<br>Resource intensive<br>Bigger footprint<br>Slow |
| NGINX | Support by Red Hat [24]<br>Widely used | Many unneeded features |
| lighttpd | Support by Red Hat [24]<br>Lightweight | Not as widely used |
| Caddy | Secure defaults<br>Simple configurations are simple to create<br>Modular | Not as widely used<br>Not as configurable<br>Hard to make more advanced configurations |
| Self-made | Small footprint<br>Fast<br>No excess features | Harder to maintain (updates are harder)<br>No support<br>Hard to add new features |

Table 14: Overview of tools considered for the web server.

In the process of selecting a suitable web server, several factors were taken into account. Some of the key aspects for finding a tool were: small footprint, ease of use, ease of maintenance, support from Red Hat and security, amongst others. Since the web server's primary task is to serve a static folder, the focus on performance was not a primary concern. The tools in contention were: Apache httpd, NGINX, lighttpd, Caddy and a simple self-made solution. A summary of the advantages and disadvantages of the different tools is presented in Table 14.

One of the main appeals of the open source Apache httpd is that it is widely used, however, their market share is dropping and is now at 20,58% [53]. This indicates that other solutions currently are more desirable than httpd, but the package is still widely supported and there is a community around the product. This helps when troubleshooting the software. Furthermore, it is part of the RHEL standard library as one of the packages supported by Red Hat [24]. In addition, Red Hat has how-to guides specifying how to deploy and configure the package on RHEL 9[35]. However, Apache httpd is a web server with many unneeded features for this project, which increases the attack surface, footprint, and resource usage and makes it slower.

As with Apache httpd, NGINX is a widely used web server. It currently has the largest market share, at 25,94% [53]. This also results in a large community and support from

[35] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/deploying_web_servers_and_reverse_proxies/setting-apache-http-server_deploying-web-servers-and-reverse-proxies, visited 10.05.2023.

Red Hat [24]. In addition, Red Hat provides some guides for installing and configuring the package[36]. Compared to Apache httpd, NGINX is a faster web server when serving static content [43]. Some of the group members are familiar with this package. Since the use case of the web server in this project is to serve a static folder, NGINX provides more features than needed. This might increase the systems attack surface.

lighttpd aims to be a secure, fast and lightweight web server[37]. However, NGINX is faster whilst using fewer resources and at the same time being more stable [61]. It is not as widely used, resulting in a smaller community. This can make it harder to troubleshoot errors. Nonetheless, the package is in the standard repository of RHEL, therefore the package is supported by Red Hat [24].

In regards to Caddy, it is one of the newer and more modular web servers on the list. It is open-source and written in Go. It has a simple management interface through files called Caddyfiles, but the Caddyfiles have a limited amount of options. In addition to the Caddyfiles, a REST API can be used to configure the web server. The API aims to have a wider range of configurable options in comparison to the Caddyfiles. Moreover, the web server aims to be secure, with features that have security in mind, such as the AutoHTTPS[38] feature that automatically generates and applies a TLS certificate. Since it is a relatively new product, the community is not that large, which by extension indicates less troubleshooting information to find online. In addition, the package is part of the Extra Packages for Enterprise Linux (EPEL) repositories, which implies that the package is not supported by Red Hat [24].

If the group had developed a web server specifically for the use case of this project, it would have had the smallest footprint since it would only include the necessary features to serve the folder containing the boot files. By extension, this would mean a smaller attack surface for the product. In addition, the group would have full control over what happens in the code. However, updates would prove a greater challenge compared to a package in the RHEL standard repository. Adding new features to the web server would also be more difficult since someone has to implement it. Another reason against creating a self-made web server is security. If a feature in the web server is no longer proven secure, a manual fix must be applied. This consideration complies with the concerns raised by the employer (see Section 6.7.1).

The initial choice of web server fell on Caddy. This was because the web server claimed good security and a simple configuration option. In addition, the group wanted to try a different web server than they were used to. However, after implementing some components in Caddy, the group encountered multiple issues, and the final choice ended up being NGINX. The primary reason for this choice was the familiarity within the group and the support from Red Hat [24]. NGINX also offered fewer security guards compared with Caddy, which was necessary to ensure the compatibility with the HTTP boot portion of the Dell UEFI implementation (see Section 7.4.3 for more details).

### 7.3.2 DNS/DHCP

In regards to the choice of DHCP server, the main desired features were support for providing boot options, ease of use and having a small footprint. A summary of the DHCP tools considered, together with their advantages and disadvantages, can be found in Table 15.

[36] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/deploying_web_servers_and_reverse_proxies/setting-up-and-configuring-nginx_deploying-web-servers-and-reverse-proxies, visited 10.05.2023.

[37] https://redmine.lighttpd.net/projects/lighttpd/wiki visited 10.05.2023.

[38] https://caddyserver.com/docs/automatic-https, visited 21.05.2023.

| Tool | Advantages | Disadvantages |
|---|---|---|
| dnsmasq | Supported by Red Hat [24]<br>Supports both DHCP and DNS<br>Simple | Not very scalable |
| ISC DHCP | Support by Red Hat [24]<br>Probably the most used DHCP server | Deprecated<br>No longer receiving support from the maintainers |
| Kea DHCP | Modern implementation of ISC DHCP | A lot of unneeded features<br>Relatively new<br>Not in standard package repository |
| dhcp from router | Already exists<br>No additional packages required<br>Centralized place for all DHCP | Does not necessarily support the correct boot options<br>Unable to provide all services through Ansible |

Table 15: Overview of tools considered for the DHCP server.

Internet Systems Consortium (ISC) has two packages for providing DHCP, ISC DHCP and ISC Kea DHCP. ISC DHCP is one of the most used DHCP servers over the last few years, however, ISC has chosen not to continue maintaining it[39] [36]. Even if ISC DHCP is deprecated, Red Hat still supports it, which makes the packages still viable [24]. The ISC DHCP server is easy to use and has a large community, making it easier to find solutions to errors. To replace it, ISC introduced Kea, which is a modern DHCP server built for scale. For instance, it uses a database as a back-end, to allow multiple instances to work together on the same network. In addition, a HTTP dashboard is provided for configuration and visualization [37].

Another option for DHCP is the dnsmasq package. It provides both DNS and DHCP in one package. It is designed to be deployed in small networks and aims to be simple to use and therefore has a small resource footprint. Additionally, this means that fewer unneeded features are implemented, something that reduces the attack surface. The package is provided through the standard RHEL repository. dnsmasq is configured through simple configuration files which are well documented. In addition, it has support for logging both DNS and DHCP queries to a log file.

Lastly, most routers include a DHCP server, and in many cases, this will be the simplest way to enable DHCP on a network. When having multiple networks, this will make it easier to get an overview of and configure the DHCP server. Using the router, no additional packages need to be installed on the servers, which helps reduce the attack surface of the system. However, not all routers have support for providing the DHCP options 60 and 67 that is needed for automatically providing the NBP location to the client. Another drawback is that when using the router as the DHCP server, Ansible is no longer the deployment method of the whole system. This might complicate the deployment, depending on the deployment environment.

| Tool | Advantages | Disadvantages |
|---|---|---|
| dnsmasq | Supported by Red Hat [24]<br>Supports both DHCP and DNS<br>Simple | Not very scalable |
| Pi-hole | Easy to use | Many unneeded features |
| Adguard home | Easy to use<br>Supports DNS-over-HTTPS | Many unneeded features |

Table 16: Overview of tools considered for the DNS server.

As with the DHCP server, simplicity and the footprint of the package were the key metrics for the choice of DNS server. The different alternatives with their advantages and disadvantages can be found in 16. dnsmasq is also part of the possible choices for DNS server. The assessment of the dnsmasq package for DNS server is the same as the one for DHCP. In addition to dnsmasq, Pi-hole and Adguard home are options that were

[39] They might however still create security updates if major vulnerabilities are discovered [36].

considered when deciding a DNS server.

Both Adguard Home and Pi-hole offer similar features. They both provide a simple web interface for managing DNS and the other provided features. The main feature of these services is to act as a DNS sinkhole and block DNS traffic to specific domains. These features are not needed for the primary use case of this system. A feature for future-proofing the system that Adguard Home provides is DNS-Over-HTTPS. This features encrypts the DNS traffic, which increases the confidentiality of the system [28]. At its core, Pi-hole uses dnsmasq to handle the DNS queries. Therefore its features set is more similar to dnsmasq[60].

Since dnsmasq both provide DNS and DHCP, in addition to being lightweight, it was chosen to serve these functions. This makes the configuration and deployment of the provisioning system simpler and lowers the system's attack surface by introducing fewer packages to the server.

### 7.3.3 Ansible Automation Platform

| Tool | Advantages | Disadvantages |
|---|---|---|
| Pushing the configuration from the orchestration service | Single source of truth<br>Simple | Difficult to handle secrets<br>Reduces the cohesion of the orchestration system |
| Running the playbooks on the clients | Easy secret management | Manual task<br>Fragile |
| Ansible Automation Platform (AAP) | Handles secrets<br>Maintained by Red Hat | Many unneeded features, increases the attack surface |

Table 17: Overview of tools and methods considered for applying the Ansible playbooks on the clients.

When deciding how to execute the Ansible playbooks on the client during the provisioning process, multiple options were considered. The different choices, together with advantages and disadvantages can be found in Table 17. The initial idea was to make the server running the orchestration service have Ansible installed and call the Ansible command from the Go code. This method would have provided a single source of truth for the scripts and reduced the complexity of the system by only having a single place of execution. However, it introduces some challenges in regard to handling secrets, where the main issue is the use of two-factor authentication. In addition, this method would reduce the cohesion of the orchestration service by giving it multiple roles and responsibilities.

To try to accommodate the issues of handling the secrets, other alternatives were considered. The second alternative aimed to use the orchestration service to transfer the Ansible playbooks to the client and have a user manually trigger the installation. This alternative solved the issue of entering the two-factor information since the user triggers the process manually. This would potentially have introduced the possibility of user error. For example, if the user would have to trigger the Ansible playbooks manually, the user could potentially alter the configuration or start the wrong playbook. Another option was to make the orchestration service start the script in the foreground of the client and make the user enter their credentials. The script would potentially have had to automatically unlock the system, open a terminal in the foreground of the screen and trigger a command. However, this could potentially introduce a fragile system because the implementation of such a system would require tight couplings to the given operative system. When components of the operative system are updated this script could easily break.

Lastly, Ansible Automation Platform (AAP) was considered. AAP is developed and maintained by Red Hat and aims to solve the problem of executing Ansible playbooks from a centralized location in an enterprise environment. A key benefit is that AAP can be integrated with secret management solutions like Hashicorp Vault, to avoid having to handle secrets manually. In addition, it includes a REST API that easily facilitates interactions with the service [63]. Even though it solves most problems that were encountered by the other solutions, it will increase the attack surface of the system because it has many features that are not strictly needed for this project. Since the deployment and configuration of AAP is a more involved process and need to be tailored to a given organization, it was decided that this would be outside the scope of this project.

### 7.3.4 Programming Language

| Tools | Advantages | Disadvantages |
|---|---|---|
| Python | Large community<br>Popular<br>Strong REST API support | Less experience<br>REST API functionality depends on external packages |
| Java | Good experience<br>Large community | Package not installed in RHEL by default<br>REST API functionality depends on external packages |
| Go | Made for web-services<br>String REST API support<br>Very good experience | Smaller community |

Table 18: List of languages considered for the orchestration service.

When developing the orchestration service, it was essential to select an appropriate programming language. The chosen language needed to meet the following criteria:

- Be actively maintained
- Be familiar to the group
- Offer robust support for REST API development
- Be supported by Red Hat

The summary of the different alternatives, with advantages and disadvantages can be found in Table 18.

A popular option that satisfies these requirements is Python[40]. Python is widely utilized and has a large community that develops libraries for the language. In conjunction with the Flask package[41], Python provides a solid foundation for creating REST APIs. Additionally, Python is supported by Red Hat [24]. However, the group has limited experience in programming web-based services, including REST APIs, in Python. Furthermore, Python's reliance on external libraries for desired functionality increases the number of dependencies and, consequently, the attack surface.

Java[42] was another language considered due to the group's experience with it. As a widely used language, Java benefits from a large community. However, Java is not installed by default on new RHEL installations, necessitating additional installation. Moreover, Java lacks strong built-in REST API support and the group has no experience in creating such services using Java.

Ultimately, Go was chosen, with which the group has significant experience in creating REST APIs. Go offers robust built-in REST API support in its standard library [21].

[40] https://www.python.org
[41] https://flask.palletsprojects.com/en/2.3.x/
[42] https://www.java.com/en/, visited 21.05.2023.

Although Go has a slightly smaller community than the other languages, which may result in more challenging troubleshooting, its advantages outweigh the disadvantages in this context.

## 7.4 Implementation

Throughout the implementation of the provisioning system, some considerations were made and challenges encountered. This section aims to discuss these considerations and challenges to give a better understanding of the implementation.

### 7.4.1 NGINX CIS Hardening

Most relevant aspects of the NGINX CIS benchmark was implemented. However, some were skipped because of compatibility reasons, or because the recommendations were not applicable to the implementation, such as recommendations regarding removing unneeded modules and self-compiling the package. A reason for this is that through Red Hat, the package is supported, providing reassurance in case an issue arises [24]. In addition, this will provide easier updates, since the updates are automatically provided through the standard package repository. For the CIS NGINX benchmark summary table, see Appendix S.

There were also some recommendations that could not be followed because of the UEFI implementation of Dell. One of these was recommendation `4.1.14`, which recommended a set of TLS ciphers to use. The client would not boot from the network boot server using this recommendation (further discussed in Section 7.4.3), therefore a modification was used. The recommended exclude filter was however still used as specified by the benchmark. The `2.1.4` recommendation that states that the `autoindex` option should be disabled, was not followed. This is because without this option the directory listing will not work. For the use case of this project, the web server acts as a HTTP file server and this option is therefore needed.

Some parts of the benchmark have been incorporated into the Ansible role intead of directly into the NGINX configuration files. This includes `2.3.1` and `2.3.2`, which specifies the file and folder owner and the access mode. These permissions are automatically set during each execution of the Ansible role. Therefore the system administrator can be certain that the permissions are correct, even if new files are added. The `4.1.1` recommendation that specifies that HTTP traffic should be redirected to HTTPS, is set through a variable to better ensure compatibility with multiple deployment environments. Another one is the `4.1.4` which dictates which TLS protocols are in use. The default variable is set to the recommended value, but the user is able to change it for better compatibility with clients.

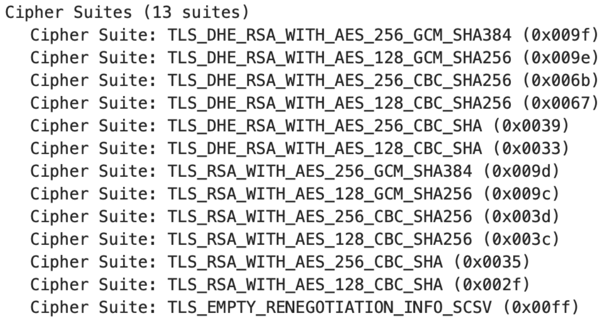
### 7.4.2 Kickstart File

Since the Kickstart file includes the same IP address and domains in multiple locations, it was desired to add these as global variables. The use of variables would make it easier for the user to change the IP address and other variables, whilst minimizing the amount of errors. However, there is no easy way of accomplishing this in a Kickstart file. First of all, there is no native method for setting variables (environment variables) outside the scope of one script location. However, there is a possibility if the script sections are run using the `--nochroot` flag to run the commands outside of the chroot jail. Using this, the variables can be contained in a file, which the sections can read and write to. This might compromise the security of the installation process and will make the Kickstart file both harder to read and update. It was therefore decided against using variables in the Kickstart files since there is no simple and clear way of doing it.

Another issue with the Kickstart file was to make it fail when for example the PRE script was unable to reach the orchestration service. In earlier versions of the Kickstart specification, the use of `exit 1` would make the installation fail, but this no longer works. One other possibility is to make the whole script section create a file on the file system when they have been executed successfully. Then, outside the script sections, one can add a `%include` statement for that file. If the file is found, the installation continues, if not, the installation fails. However, this is an unsophisticated method of doing it. Again, the script section would have to be run using the `--nochroot` flag, which might compromise the security and the integrity of the installation. However, in RHEL 9, it is possible to use the `--erroronfail` flag when defining the script to make the installation stop when an error occurs [25]. The use of this flag should be considered when creating the Kickstart file for a new configuration.

### 7.4.3 Alternative Web Server

As stated in Section 7.3, the first iteration of the provisioning system relied on Caddy as the web server. It was however, changed to NGINX at a later stage because of some issues that occurred. The main issue was the lack of support for TLS ciphers on the Dell client. There is no list provided by Dell regarding which ciphers they support, but in the TLS1.2 client hello packet of the initial TLS handshake, it is possible to see the preferred cipher suites for the client (see Figure 32). Since many of these ciphers are now considered unsafe, Caddy does not support them[43] [44] [9]. Since the server and the client did not have a common cipher suite, a connection could not be established over HTTPS.



```
Cipher Suites (13 suites)
   Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
   Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
   Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x006b)
   Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x0067)
   Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
   Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
   Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
   Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
   Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
   Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
   Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
   Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
   Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
```

Figure 32: The cipher suites supported by the Dell UEFI HTTPS boot client, visualized in Wireshark.

In addition to the TLS cipher suite issue, many issues occurred when trying to extract the access logs properly. Even if the specified log folder was owned by the caddy user and the file and folder permissions were set to read, write and execute for all users, the log file was not created because of a permission denied error. This issue was never resolved by the group. Since having insight into the application through logs is vital for security, this would have caused a major security flaw in the provisioning system. Another issue was the fact that Caddy was part of the Extra Packages for Enterprise Linux (EPEL) library, instead of the RHEL standard library (as alluded to in Section 6.7.1). This could have caused compatibility and support issues in the future.

These issues, in conjunction with flawed documentation, a relatively small community, and feedback from the employer, made the group decide to switch to NGINX instead (see Section 6.7.1). Using NGINX made the deployment more flexible, since Caddy did not provide many configuration options. However, Caddy justifies this by this making the web server more secure, but in the case of the provisioning system, it caused some compatibility issues.

[43] See https://caddyserver.com/docs/caddyfile/directives/tls#ciphers, visited 12.05.2023, for the list of supported ciphers in Caddy.

```
{% if log_file_location is defined %}
log-facility= {{ log_file_location }}
{% endif %}
```

Listing 5: The jinja2 code in the Ansible template that makes the log-facility option optional.

### 7.4.4 Ansible Deployment Playbooks

The group decided to split the Ansible roles for DNS and DHCP to make it easier to only deploy one of the components at the time. In addition, this would increase the cohesion of the roles. However, this introduced some complexity into the roles because dnsmasq has some global options which can only be specified once in any of its configuration files. For instance, if the `log-facility` configuration option is defined in two files, dnsmasq will not start. Since the roles are supposed to be able to function independently of each other, both roles had to define this option. The solution to this issue was to make the log option optional (as shown in Listing 5). This lets the user decide which role should enable the logging. However, this solution is not ideal since it permits the system administrator to not have logging enabled. This has been specified in the documentation, to make the user aware of this issue.

The presence of the GRUB Network Boot Program (NBP) and the orchestrator executable file within the Ansible repositories complicates their update process. If a new version is implemented, it is a manual process of uploading the file to the repositories. A simpler solution for the orchestration service would be to deliver the executable through a local package repository. This would make it easier to provide updates to the endpoints and make the Ansible playbook run faster. In addition, it might help avoid mismatches between the newest version of the executable and the one located in the Ansible playbook repository. When it comes to the delivery of the GRUB NBP, it is currently handled by the Ansible `ansible.builtin.copy` module which copies the files from a folder located in the repository. This is a slow approach because there are many files to be copied. Furthermore, the host needs to have UEFI enabled, which consequently means that the files cannot be generated on every host. Not all servers will have UEFI enabled, therefore it is easier and more consistent to deliver the GRUB NBP through Ansible.

### 7.4.5 The Orchestration Service

The development process began by creating individual packages, such as the ”requests” and ”clients” packages, which would be utilized by the other packages. This approach led to some unforeseen challenges, which necessitated some modifications in the package structure.

The service includes both a configuration package and a logging package. However, these two packages are dependent on each other. The logging package relies on the configuration package being initialised and retrieving parameters needed for logging initialisation, while the configuration package requires the logging package for proper logging. The solution was to have the configuration package use the Go standard logging instead of the custom logging, which is not optimal. These logs are not persisted to the file, only printed to the console.

During the provisioning process, a client's state updates are logged in the orchestration service's memory and concurrently written to a log file. This state history can be accessed via the service's API, provided that the service instance remains operational. However, following a restart, the service loses this history from its memory and cannot provide the information through the API. Consequently, the only remaining method to access these logs involves retrieving the pertinent log file and manually searching through it. Generating post-provisioning reports and historical installation statistics are two recommended actions for further work. Such installation statistics may encompass average installation duration, the ratio of successful to failed installations, and other relevant metrics.

In Go, active error-handling management is essential due to the language's distinct error-handling approach compared to other languages[15]. Two primary principles were applied: log errors as early as possible and return errors as far up the call stack as possible. These principles ensure that no error messages remain unhandled and reduce the number of functions that deal with errors, as only top-level functions need to determine what actions to take based on errors. Early logging, however, allows for better visibility of what went wrong and where it occurred in the service.

Error handling is also crucial in testing. When writing unit tests, each program function must be tested, including return values and errors. It is necessary to have a consistent way of referring to errors, allowing unit tests to compare the returned errors. In this service, constants are used to address this issue. Functions that return errors, when possible, return a constant containing the correct error. Constants are defined for each package, enabling respective tests to refer to the same constant and check for expected errors in various scenarios.

When creating APIs, it is vital to provide clear and concise error messages so that users understand what went wrong. This has mostly been implemented, but a challenge arose with the Ansible Automation Platform (AAP) package. The aap package uses AAP's API to communicate with the instance, and when AAP returns an error, the error message is in the HTTP response body. However, the package does not extract this message and deliver it to the user, resulting in more generic feedback for the user.

## 7.5 Security Assessment

This section will elaborate and discuss the process of conducting the security assessment, focusing on the challenges that were faced throughout the process. The group held meetings with a Security Architect (SARC) from the employer before starting the security assessment. This provided valuable insight into the employer's procedures and framework, and how they address various issues. Since the group had not conducted a security assessment before, this information was extremely valuable. The group was able to ask initial questions, as well as understand how the framework operates and should be utilized. During the security assessment, the group had a second meeting with the SARC, who provided feedback and advice. This proved to be exceptionally helpful as the group had several challenges during the assessment.

Conducting the BIA presented several challenges. Given the group's inexperience with executing a BIA, considerable time was spent defining the scope of the confidentiality, integrity, and availability (CIA) assessment. There was uncertainty about whether the CIA assessment should encompass only the directly affected systems, or also include systems that could be indirectly impacted by a problem with the provisioning process. Following a meeting with the SARC, it was determined that the assessment needed to evaluate potential effects on the organisation as a whole. Furthermore, defining the informational assets also sparked some discussion.

Some problems were faced while performing the threat assessment as well. The assessment uses several relevant sources that describe the threat level for the environment in which the system will be deployed. For example, the threat and risk reports from some secret services in Norway. However, the vast landscape of intelligence information contains numerous closed sources. Notable sources are FireEye[44], Mandiant[45], Crowdstrike[46], and BAE-systems[47], which all operate with reports that are not open to the public. Consequently, the group generally lacks access to these reports, and if the group

[44] https://www.trellix.com/en-us/index.html, visited 21.05.2023.

[45] https://www.mandiant.com, visited 21.05.2023.

[46] https://www.crowdstrike.com, visited 21.05.2023.

[47] https://www.baesystems.com/en/home, visited 21.05.2023.

had access, the closed nature of the sources meant that they could not be utilized in the assessment. Furthermore, each of the different organisations that would consider deploying the provisioning system in their environment, have a larger and more specific intelligence information base to base their assessment on. This information is often sensitive, and could not have been used in the threat assessment in this thesis.

Furthermore, identifying and defining the scope of risks for the risk treatment and evaluation part of the assessment was a significant challenge. This task was particularly difficult when the risks had to be evaluated with the employer's critical function in mind. However, with guidance from the SARC, the group was able to pinpoint the relevant scope, including risks and assets, and how to describe them effectively. Another challenge was to assess the severity of each risk, which required an understanding of not only the potential vulnerabilities and consequences, but also the employer's existing control measures and how they would affect the severity. These challenges expressed the importance of fully understanding the environment and context when conducting a security assessment.

### 7.5.1 Evaluation

The security assessment was evaluated by a SARC, as mentioned in Section 6.7. This evaluation also consisted of recommendations for improvement and specificity. Firstly, it was recommended to look at the OWASP definition of threat modelling[48], regarding the threat assessment. This definition, in combination with other threat modelling tools, is utilized by SARCs, in collaboration with threat analytics, to understand where and how different tactics, techniques, and procedures (TTP) can exploit the system vulnerabilities and lead to consequences. Secondly, it was suggested that it might be relevant to highlight the inherent uncertainty in the threat level, given that the threat is directed towards a support system, rather than the critical systems of the employer.

Thirdly, it was recommended to clarify in the introduction of the security risk assessment what the scope is, in addition to specifying the environment into which the provisioning system is being introduced. Lastly, it was advised to include risk before and after measures in the summary, along with a list of the most highly recommended countermeasures to implement in order to minimize risks. In conclusion, the evaluation did not indicate anything that required change or was done incorrectly, only recommendations for improving the assessment and ensuring it includes all necessary aspects for thorough understanding. These are improvements that should be conducted when working further on this project. The full evaluation can be found in Appendix G.

## 7.6 User Testing

The group carried out qualitative user tests as part of the quality assurance process, as described in Section 6.7. These user tests served as an expert evaluation of the automatic provisioning system. The group considered performing user tests on other candidates outside the employer's IT department, to examine the usability and efficiency of the automatic provisioning system. It was concluded, since other candidates lacked background knowledge regarding the context of the automatic provisioning system and provisioning in general, that the results of these user tests would not be helpful.

One point of discussion is whether the system should have been tested earlier by experts, rather than after completion, to not solely rely on tests which confirmed the product's quality. The reason this was not carried out earlier is that through feedback from the architects during the development phase (see Section 6.7), the group received valuable insights about the architecture and implementation. The system was also created in consultation with the employer, who ultimately represents the end-user of the automatic

[48] https://owasp.org/www-community/Threat_Modeling, visited 16.05.2023.

provisioning system. Therefore, the group saw no value in having similar user tests earlier in the process, nor performing user tests on other users besides the few who would actually use the system.

However, another point to note is that the group received minor feedback and suggestions concerning documentation and small additional features. There was no feedback on the architecture or implementation, which, although not the focus of the user tests, was reassuring as the tests were conducted by professionals in the field. If the user tests had been conducted earlier, the group could have implemented the minor changes and conducted another round of tests to confirm improvements. However, due to time constraints, this was not possible.

## 7.7 Project Process

In this section, the group's experience concerning the project process is evaluated and discussed. This includes the initial project planning, meetings with both the employer and academic supervisor and collaboration within the group, along with the evaluation of the development model.

### 7.7.1 Planning

Looking back, the group spent more time than anticipated on planning the project. Most parts of the first month was dedicated to planning, and in hindsight, the group believes this time could have been used more efficiently. It was also challenging to specify the different requirements for the provisioning system early in the project, as the group lacked knowledge regarding how a provisioning system could function. This uncertainty made it difficult to establish a clear direction for the project at this early stage. However, having a precise and detailed project plan did provide a structured and clear expectations, as well as a strong foundation for the project's development and execution. For example, deciding which meetings required minutes, and establishing a clear format for them, contributed to maintaining consistency and structure throughout the project (see Appendix M for examples of meeting minutes).

### 7.7.2 Meetings

The group held regular meetings with both the employer and the academic supervisor. The discussions with the employer primarily focused on technical aspects throughout the majority of the project. The employer contributed valuable insights, knowledge, and feedback concerning the project's technical components. As the project advanced and the focus shifted towards writing the thesis, the necessity of meetings with the employer decreased. Despite this, these meetings still offered an opportunity to discuss the progression of the thesis and any concerns related to its completion and submission. These meetings proved to be very valuable to the group.

Similarly, the meetings with the academic supervisor, Kiran Raja, were very valuable. These meetings focused on academic writing and the delivery of the thesis report. As this was the first time the group undertook a project of such a large scale, having Kiran Raja as a supervisor greatly assisted in discussing report structure, content and quality. Occasionally, there were misunderstandings regarding technical aspects, causing some back-and-forth in decision-making. However, these misunderstandings had minimal impact on the overall project progress and thesis writing.

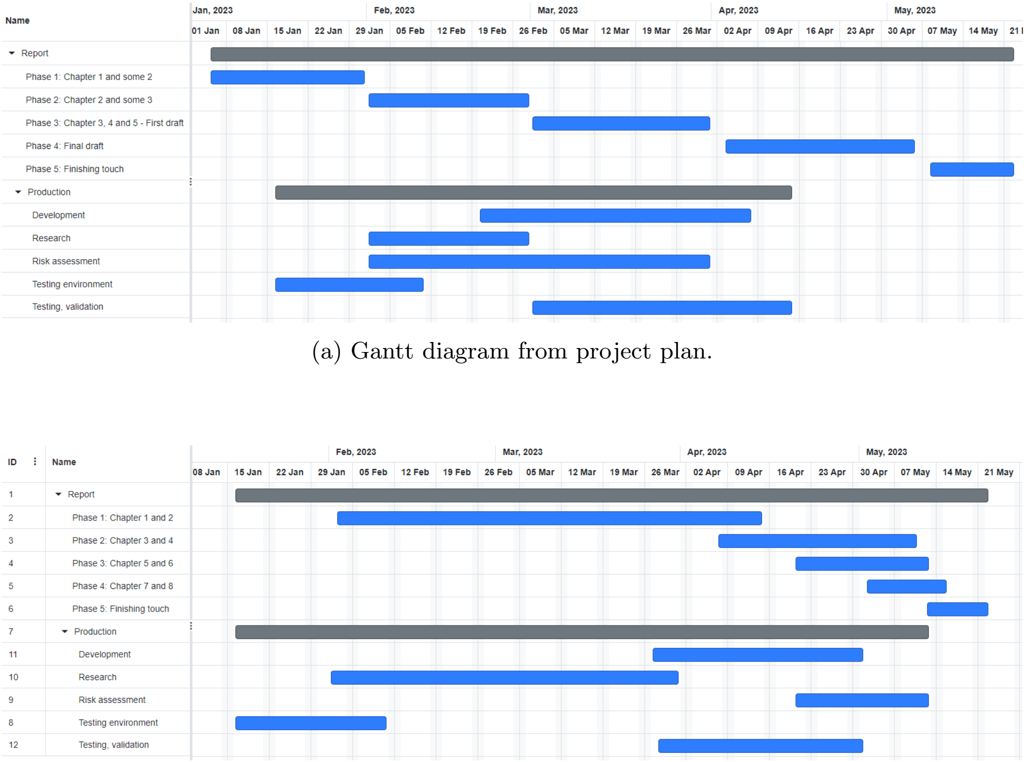
### 7.7.3 Collaboration

Collaboration within the group went smoothly for the most part, but there were times when the group became interdependent, causing some tasks to be delayed longer than necessary. As a result, different aspects of the project were more time-consuming than planned. However, the group maintained the mentality that all members should stay up-to-date on the progress, which resulted in sufficient collaboration and communication flow.

### 7.7.4 Development Process

In general, the group perceived the development model as well-suited for this project. Its iterative and adaptable approach helped seamless modifications to the execution of the project when necessary. Regarding the execution of the project, certain aspects need consideration. Firstly, a substantial amount of time was allocated to internal meetings, such as status meetings (from the Scrum framework), which occasionally seemed excessive since the group consistently communicated throughout the day. Despite this, the internal meetings proved beneficial in the long term. They ensured that the group did not become overwhelmed with the workload, as they provided an overview of the project by promoting communication regarding progress and concerns.

Secondly, different parts of the project took longer than initially anticipated. As mentioned in Section 3.3.1, the Gantt chart needed to be revised. However, the actual execution of the project did not align with the group's expectations after the revision of the Gantt chart (for more information, refer to Section 7.7.5). The real-life execution of the project differed from the plan. Figure 33 shows how the project actually progressed, compared to the initial Gantt diagram from the project plan. The production phase ended on the 30th of April, including all the technical aspects of the project, such as development and unit testing. The security assessment was successfully finalised by the 10th of May, followed by the completion of expert evaluations by the 14th of May. The group managed to prepare the first draft of the bachelor thesis by the 15th of May. See Appendix T for an overview of the different versions of the Gantt diagram for this project.



(a) Gantt diagram from project plan.

(b) Actual execution of project.

Figure 33: Initial vs Final Gantt Diagram.

Nonetheless, employing this development model simplified the process of changing the projected timeline. Moreover, the iterative approach was beneficial when changes to the duration of each sprint were necessary. As outlined in Section 3.3.1, the initial length of each sprint was set to 10 days. The group modified these based on the specific needs at the time. This proved advantageous, as the sprints had distinct goals and tasks, and thus needed varying lengths.

Lastly, the group experienced that they created broad issues, which subsequently needed to be transferred over to the following sprint, causing the work progression to deviate from initial expectations. Through these experiences, the group learned to specify issues more precisely and to create multiple smaller issues to ensure the sprint goal was met. Apart from this, the group found the use of issues to be highly beneficial, as it provided a clear overview of the project's tasks and made it easier to manage information related to these tasks.

In summary, using Kanban board and Scrum, made it easier for the group to divide the work. The iterative development method allowed the group to improve the development process as the project progressed. The iterative approach, together with strong communication and time management, made the project's execution manageable.

### 7.7.5 Time Management

Given the significant scale of the project, efficient time management was crucial. As the project neared its completion, the group felt the pressure of an increasing workload. Looking back, the group fell short by about two weeks in terms of total work hours. Though two weeks might seem trivial, it accounts for roughly 200-230 missed working hours in total. There could be various reasons for this. For instance, the group could have prioritized differently in the initial months so that more time was devoted to the project. However, this would have impacted the work with other courses the group was managing concurrently with the bachelor thesis. Additionally, one group member logged more hours than the others[49], leading to an uneven distribution of work hours (see Figure 34). This could be because of the lack of a formal procedure for tracking work hours, or variances in individual working schedules, which ultimately resulted in some members putting in more time over the course of the project. For a complete and detailed summary of the time spent on the project, refer to Appendix U.

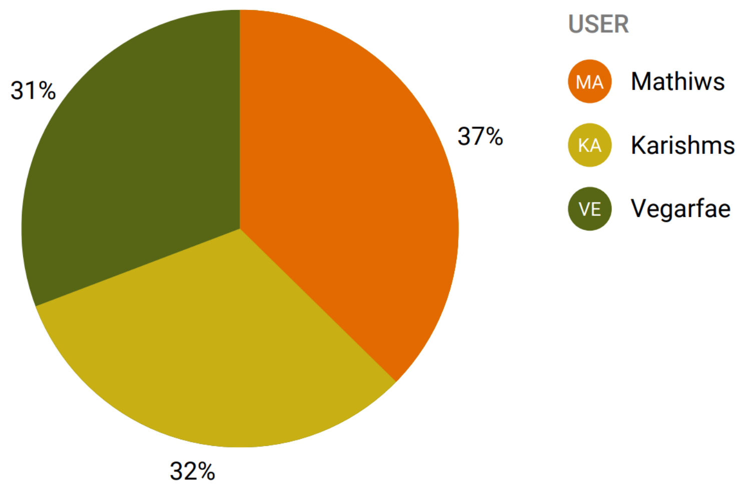


Figure 34: Pie chart showing total working hours in a project, divided by group member from Toggl

As shown in Figure 35, more than half of the project's working hours were spent on documentation. This included both the security assessment and the writing of the bachelor's thesis, making it understandable that this took up a significant portion of the project's time. In addition, the majority of the thesis writing was postponed until the start of May, which led to a period of intense writing for the group, rather than spreading it out over the entire project. However, if the documentation label was divided into thesis writing and security assessment, the overview of time spent would have been more clear.

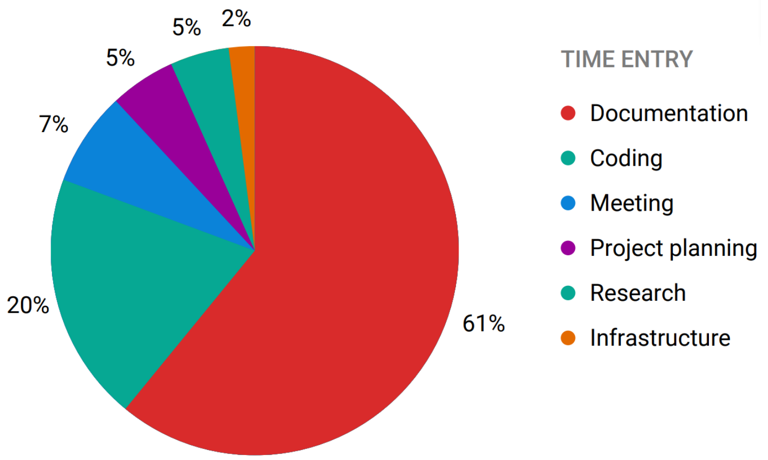


Figure 35: Pie chart showing total working hours, divided by label from Toggl.

[49] The time-tracking tool Toggl was utilized to provide summary reports. https://toggl.com/, visited 21.05.2021.

# 8 Conclusion

Throughout this project, a provisioning system for physical clients has been developed, designed to meet the security requirements for deployment in environments with low risk acceptance. This was accomplished by integrating UEFI HTTP boot, Ansible, DHCP, DNS, a web server, and a self-developed orchestration service. Evaluation through a security assessment, user tests, and employer feedback demonstrated that the resulting solution is automated, scalable, maintainable, and more standardised and efficient compared to the existing solution.

## 8.1 Achieved Goals

The solution aims to fulfil the result and effect goals found in Section 1.3. The result goals were achieved by the group successfully developing an automatic and secure provisioning system. It focused on promoting consistency and eliminating manual processes prone to human error, in contrast to a manual process. The solution has been developed utilizing the SSDLC checklist and assessed using the security assessment framework from the employer. An evaluation of the group's security assessment, confirms that it meets the expectations of the employer. In addition, the solution offers scalability to accommodate multiple clients needing provisioning. The solution is not fully automated, but compared to the current process, the amount of human interaction has been significantly reduced as confirmed by the user tests. Furthermore, the user tests suggest a significant time reduction in the provisioning process compared to the manual process.

The effect goals cannot be confirmed nor denied. However, the results of the project indicate that the solution is aligned with the intended goals. The qualitative user tests provide evidence that the system reduces both the time and the need for human interaction during provisioning. While the group lacks concrete quantitative statistics, the experiences of the user test candidates, and comparisons with the current process, support this conclusion. In addition, since the clients are provisioned more consistently, the maintenance and management of clients will be easier. Moreover, the group has taken multiple steps to ensure the usability and scalability of the provisioning system, such as adhering to best practices for coding and development. Furthermore, with the entire solution defined in code and with limited self-developed components, maintenance becomes more straightforward. The solution was thoroughly security assessed, with a resulting risk level within the risk acceptance criteria.

## 8.2 Further Work

In the following list, the potential directions for future research and development within the context of this thesis will be highlighted. It aims to identify areas where the most important improvements, refinements, or expansions could be made to the current solution. By addressing these aspects, it is anticipated that this section will serve as a guide for future maintainers, developers, researchers, and practitioners seeking to build upon the findings and contributions of this thesis.

- An extensive security assessment tailored to the specific requirements of the individual environment and the information base of the individual organisation should be conducted before deployment in other environments with low risk acceptance.
- Thee security assessment should be modified to reflect the feedback given from the employer evaluation.
- For deployment in environments with weak physical access control, the implementation of supplementary logical access controls should be considered.

- Testing the provisioning of multiple clients concurrently should be conducted, as these were not conducted during this project. Conducting such tests before deployment is crucial, particularly in environments where this functionality may be critical during acute situations.

- Signing of the boot programs should be conducted, so that secure boot can be enabled on supported clients.

- Implementing DNS over HTTPS[50] would decrease the risks associated with utilizing a DNS server in the solution.

- The solution documentation must be further improved, to ensure usability and maintainability for individuals not familiar with the project's concepts. This includes the user guide, admin guide, and the README of the orchestration service.

- The logging mechanism for capturing errors during the process requires improvement, particularly in light of the client's inability to report errors occurring during the Operating System installation.

- Alternatives for further automation of the deployment should be evaluated. Most prominent is the manual process of moving the GRUB menu files and the orchestration service's executable to the deployment repository.

[50] https://www.rfc-editor.org/rfc/rfc8484, visited 21.05.2023.

# References

[1] *1.3. Supported Usage*. Red Hat. URL: https://access.redhat.com/documentation/en-us/red_hat_satellite/6.4/html/planning_for_red_hat_satellite_6/chap-red_hat_satellite-architecture_guide-introduction_to_red_hat_satellite#sect-Red_Hat_Satellite-Architecture_Guide-Red_Hat_Satellite_6_Supported_Usage (visited on 20/04/2023).

[2] *7.7. Implementing PXE-less Discovery*. Red Hat. URL: https://access.redhat.com/documentation/en-us/red_hat_satellite/6.10/html/provisioning_guide/configuring-the-discovery-service#implementing-pxe-less-discovery (visited on 20/04/2023).

[3] *Ansible Documentation*. Red Hat, Mar. 2023. URL: https://docs.ansible.com/ansible/latest/index.html (visited on 13/04/2023).

[4] *Automation Controller User Guide v4.3*. Red Hat. URL: https://docs.ansible.com/automation-controller/latest/html/userguide/index.html (visited on 13/04/2023).

[5] baeldung. *How to Install and Use Popular Linux Bootloaders*. Oct. 2022. URL: https://www.baeldung.com/linux/popular-bootloaders (visited on 20/05/2023).

[6] *Business Impact Analysis*. Ready Campaign (U.S. Department of Homeland Security), June 2021. URL: https://www.ready.gov/business-impact-analysis (visited on 25/04/2023).

[7] *Chapter 1. Installation methods*. Red Hat. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html/performing_a_standard_rhel_9_installation/installation-methods-advanced_installing-rhel#installation-methods_advanced_installation-methods (visited on 17/04/2023).

[8] *Chapter 1. Introduction to Red Hat Satellite*. Red Hat. URL: https://access.redhat.com/documentation/en-us/red_hat_satellite/6.12/html/satellite_overview_concepts_and_deployment_considerations/introduction_to_server_planning (visited on 20/04/2023).

[9] Cloudflare. *Compliance status*. URL: https://developers.cloudflare.com/ssl/reference/cipher-suites/compliance-status/ (visited on 14/05/2023).

[10] *Conduct Business Impact Analysis*. The European Union Agency for Cybersecurity (ENISA). URL: https://www.enisa.europa.eu/topics/risk-management/current-risk/bcm-resilience/bi-analysis (visited on 25/04/2023).

[11] Stan Cox. *How to use Secure Boot to validate startup software*. June 2022. URL: https://www.redhat.com/sysadmin/secure-boot-systemtap (visited on 12/05/2023).

[12] Neo Cui. *Using HTTP Boot to Install an Operating System on Lenovo ThinkSystem servers*. May 2019. URL: https://lenovopress.lenovo.com/lp0736.pdf (visited on 30/04/2023).

[13] Delia. *How to network boot multiple computers within the LAN?* Dec. 2022. URL: https://www.ubackup.com/articles/network-boot-multiple-computers-1004.html#toc.0.4958573748006663 (visited on 12/04/2023).

[14] *dnsmasq*. Debian. URL: https://wiki.debian.org/dnsmasq (visited on 13/04/2023).

[15] *Error handling and Go*. URL: https://go.dev/blog/error-handling-and-go (visited on 21/05/2023).

[16] Damon Garn. *Static and dynamic IP address configurations for DHCP*. Red Hat, May 2021. URL: https://www.redhat.com/sysadmin/static-dynamic-ip-1 (visited on 13/04/2023).

[17] Owen Garrett. *Using NGINX and NGINX Plus with SELinux*. Aug. 2018. URL: https://www.nginx.com/blog/using-nginx-plus-with-selinux/ (visited on 29/04/2023).

[18] GitLab. *Dependency Scanning*. URL: https://docs.gitlab.com/ee/user/application_security/dependency_scanning/ (visited on 01/05/2023).

[19] GitLab. *Secret Detection*. URL: https://docs.gitlab.com/ee/user/application_security/secret_detection/ (visited on 01/05/2023).

[20] GitLab. *Static Application Security Testing (SAST)*. URL: https://docs.gitlab.com/ee/user/application_security/sast/ (visited on 01/05/2023).

[21] *Go, net/http documentation*. URL: https://pkg.go.dev/net/http (visited on 21/05/2023).

[22] Red Hat. *2.4.4. Changing port numbers*. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/managing_confined_services/sect-managing_confined_services-configuration_examples-changing_port_numbers (visited on 16/05/2023).

[23] Red Hat. *3.3. Preparing Installation Sources*. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/installation_guide/sect-making-media-additional-sources#sect-making-media-sources-network (visited on 17/04/2023).

[24] Red Hat. *Package manifest*. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/package_manifest/index (visited on 01/05/2023).

[25] Red Hat. *Performing an advanced RHEL 9 installation*. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/9/html-single/performing_an_advanced_rhel_9_installation/index (visited on 12/05/2023).

[26] Red Hat. *Red Hat Enterprise Linux Life Cycle*. URL: https://access.redhat.com/support/policy/updates/errata/ (visited on 28/04/2023).

[27] Red Hat. *What is a REST API?* Aug. 2020. URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (visited on 30/04/2023).

[28] AdGuard Home. *AdGuard Home*. Feb. 2023. URL: https://github.com/AdguardTeam/AdGuardHome/blob/master/README.md (visited on 10/05/2023).

[29] *How Ansible works*. Red Hat. URL: https://www.ansible.com/overview/how-ansible-works (visited on 13/04/2023).

[30] *https://learn.microsoft.com/en-us/mem/autopilot/*. Microsoft. URL: https://www.microsoft.com/cms/api/am/binary/RE4yfB2 (visited on 25/04/2023).

[31] IBM. *Linux initial RAM disk (initrd) overview*. July 2006. URL: https://developer.ibm.com/articles/l-initrd/ (visited on 20/05/2023).

[32] ICANN. *DNSSEC – What Is It and Why Is It Important?* URL: https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en (visited on 30/04/2023).

[33] Incognito. *DHCP Options in Plain English*. July 2021. URL: https://www.incognito.com/tutorials/dhcp-options-in-plain-english/ (visited on 30/04/2023).

[34] Center for Internet Security. *CIS Benchmarks List*. URL: https://www.cisecurity.org/cis-benchmarks (visited on 08/05/2023).

[35] *Introducing Discovery as a Reporting Tool*. Red Hat. URL: https://access.redhat.com/articles/4605801 (visited on 20/04/2023).

[36] ISC. *ISC DHCP*. URL: https://www.isc.org/dhcp/ (visited on 17/05/2023).

[37] ISC. *Kea DHCP*. URL: https://www.isc.org/kea/ (visited on 17/05/2023).

[38] *Juju OLM Documentation*. Canonical. URL: https://juju.is/docs/olm (visited on 25/04/2023).

[39] *Juju SDK Documentation*. Canonical. URL: https://juju.is/docs/sdk (visited on 25/04/2023).

[40] Simon Kelley. *Dnsmasq*. URL: https://thekelleys.org.uk/dnsmasq/doc.html (visited on 13/04/2023).

[41] *Kickstart files for Red Hat Enterprise Linux*. IBM Cooperation, Oct. 2022. URL: https://www.ibm.com/docs/en/linux-on-systems?topic=methods-kickstart-files-red-hat-enterprise-linux (visited on 17/04/2023).

[42] Henrik Kniberg and Mattias Skarin. *Kanban and Scrum-making the most of both*. Visited 12-04-2023. Lulu. com, 2010.

[43] Abhimanyu Krishnan. *NGINX vs Apache: Head to Head Comparison.* Dec. 2022. URL: https://hackr.io/blog/nginx-vs-apache (visited on 10/05/2023).

[44] SSL Labs. *SSL and TLS Deployment Best Practices.* Jan. 2020. URL: https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices (visited on 14/05/2023).

[45] Ansible Lint. *Ansible Lint Documentation.* URL: https://ansible-lint.readthedocs.io/ (visited on 01/05/2023).

[46] *MAAS.* Canonical, Apr. 2023. URL: https://maas.io/docs/about-maas (visited on 24/04/2023).

[47] Caeleigh MacNeil. *What is a business impact analysis (BIA)? 4 steps to prepare for anything.* Dec. 2022. URL: https://asana.com/resources/business-impact-analysis (visited on 26/04/2023).

[48] *Microsoft Intune securely manages identities, manages apps, and manages devices.* Microsoft, Apr. 2023. URL: https://learn.microsoft.com/en-us/mem/intune/fundamentals/what-is-intune (visited on 20/04/2023).

[49] Kief Morris. 'Patterns and Antipatterns for Structuring Stacks'. In: *Infrastructure as code: Dynamic Systems for the cloud age.* 2nd. O'Reilly Media, Inc., 2021.

[50] Mozialla. *Content Security Policy (CSP).* URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP (visited on 29/04/2023).

[51] Mozialla. *X-Frame-Options.* URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options (visited on 29/04/2023).

[52] Nick Naziridis. *Guide to TLS Standards Compliance.* Feb. 2021. URL: https://www.ssl.com/guide/tls-standards-compliance/ (visited on 01/05/2023).

[53] netcraft. *April 2023 Web Server Survey.* Apr. 2023. URL: https://news.netcraft.com/archives/2023/04/27/april-2023-web-server-survey.html (visited on 10/05/2023).

[54] NGINX. *nginx documentation.* URL: https://nginx.org/en/docs/ (visited on 17/04/2023).

[55] NGINX. *Serving Static Content.* URL: https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/ (visited on 01/05/2023).

[56] NGINX. *What is HTTP/2?* URL: https://www.nginx.com/resources/glossary/http2/ (visited on 29/04/2023).

[57] NTNU. *Informasjonssikkerhet - risikovurdering.* URL: https://i.ntnu.no/wiki/-/wiki/Norsk/Informasjonssikkerhet+-+risikovurdering (visited on 20/05/2023).

[58] Oracle. *dnsmasq (8).* July 2022. URL: https://docs.oracle.com/cd/E88353_01/html/E72487/dnsmasq-8.html (visited on 30/04/2023).

[59] OWASP. *HTTP Strict Transport Security Cheat Sheet.* URL: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html (visited on 29/04/2023).

[60] Pi-hole. *Pi-hole Documentation.* URL: https://docs.pi-hole.net/ftldns/ (visited on 17/05/2023).

[61] Jeff Poyzner. *Node vs Apache vs Lighttpd vs Nginx.* Mar. 2016. URL: https://www.linkedin.com/pulse/node-vs-apache-lighttpd-nginx-jeff-poyzner/ (visited on 10/05/2023).

[62] Michael Preuss. *Storage and Transfer limits.* URL: https://about.gitlab.com/pricing/faq-paid-storage-transfer/ (visited on 17/05/2023).

[63] *Red Hat Ansible Automation Platform.* Red Hat. URL: https://www.redhat.com/en/technologies/management/ansible#:~:text=Ansible\%20Automation\%20Platform\%20provides\%20an,to\%20security\%20and\%20network\%20teams (visited on 13/04/2023).

[64] Bob Reselman. *Pull vs. push in automated VM provisioning: What you need to know.* Red Hat, Oct. 2021. URL: https://www.redhat.com/architect/pull-push-provisioning-cicd (visited on 13/04/2023).

[65] Snyk. *Secure Software Development Lifecycle (SSDLC)*. URL: https://snyk.io/learn/secure-sdlc/ (visited on 07/05/2023).

[66] SUSE. *Setting up a UEFI HTTP Boot Server*. URL: https://documentation.suse.com/sles/15-SP2/html/SLES-all/cha-deployment-prep-uefi-httpboot.html (visited on 17/04/2023).

[67] Teams. *Homepage Teams*. URL: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software (visited on 26/01/2023).

[68] S.A. Torabi, H. Rezaei Soufi and Navid Sahebjamnia. 'A new framework for business impact analysis in business continuity management (with a case study)'. In: *Safety Science* 68 (2014), pp. 309–323. ISSN: 0925-7535. DOI: https://doi.org/10.1016/j.ssci.2014.04.017. URL: https://www.sciencedirect.com/science/article/pii/S0925753514001027.

[69] Michelle Tsai. *Using HTTPS Boot to Install an Operating System on Lenovo ThinkSystem Servers*. Apr. 2022. URL: https://lenovopress.lenovo.com/lp1584.pdf (visited on 25/04/2023).

[70] UEFI. *UEFI FAQS*. URL: https://uefi.org/faq (visited on 17/04/2023).

[71] *What are Kickstart installations*. CentOS. URL: https://docs.centos.org/en-US/8-docs/advanced-install/assembly_kickstart-installation-basics/ (visited on 17/04/2023).

[72] *What is an Ansible Playbook?* Red Hat, May 2022. URL: https://www.redhat.com/en/topics/automation/what-is-an-ansible-playbook#:~:text=Ansible\%20Playbooks\%20are\%20lists\%20of,as\%20which\%20user\%20executes\%20it. (visited on 13/04/2023).

[73] *What is Infrastructure as Code (IaC)?* Red Hat, May 2022. URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac (visited on 13/04/2023).

[74] *What is juju*. Canonical. URL: https://juju.is/#what-is-juju (visited on 25/04/2023).

[75] *What is SELinux?* Red Hat, Aug. 2019. URL: https://www.redhat.com/en/topics/linux/what-is-selinux (visited on 12/04/2023).

[76] *Windows Autopilot documentation*. Microsoft. URL: https://learn.microsoft.com/en-us/mem/autopilot/ (visited on 25/04/2023).

[77] *Windows Autopilot for pre-provisioned deployment (Public preview)*. Microsoft. URL: https://learn.microsoft.com/en-us/mem/autopilot/pre-provision (visited on 25/04/2023).

[78] *Windows Autopilot Reset*. Microsoft. URL: https://learn.microsoft.com/en-us/mem/autopilot/windows-autopilot-reset (visited on 25/04/2023).

[79] *Windows Autopilot self-deploying mode (Public Preview)*. Microsoft. URL: https://learn.microsoft.com/en-us/mem/autopilot/self-deploying (visited on 25/04/2023).

[80] *Windows Autopilot user-driven mode*. Microsoft. URL: https://learn.microsoft.com/en-us/mem/autopilot/user-driven (visited on 25/04/2023).

# Appendix

## A Introduction to the Employer

**This Appendix has been redacted from the public version of this thesis.**

# B Justifications of BIA

**This Appendix has been redacted from the public version of this thesis.**

# C Business Impact Assessment (BIA)

**This Appendix has been redacted from the public version of this thesis.**

# D Threat Assessment

**This Appendix has been redacted from the public version of this thesis.**

# E Security Risk Assessment

**This Appendix has been redacted from the public version of this thesis.**

# F Deployment at the Employer

**This Appendix has been redacted from the public version of this thesis.**

# G Security Assessment Evaluation

**This Appendix has been redacted from the public version of this thesis.**

# H Secure System Development Life Cycle (SSDLC) Checklist

**This Appendix has been redacted from the public version of this thesis.**

# I Employer Pipeline

**This Appendix has been redacted from the public version of this thesis.**

# J User Testing

**This Appendix has been redacted from the public version of this thesis.**

# K Project Plan

**This Appendix has been redacted from the public version of this thesis.**

# L Project Contract

**This Appendix has been redacted from the public version of this thesis.**

# M Meeting Minutes

**This Appendix has been redacted from the public version of this thesis.**

# N NGINX Example Configurations

```
######## /etc/nging/nginx.conf ########
# Managed by Ansible

user nginx;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 1024;
}

http {
        ##
        # Basic Settings
        ##
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 10;
        types_hash_max_size 4096;

        limit_conn_zone $binary_remote_addr zone=limitperip:10m;
        limit_req_zone $binary_remote_addr zone=ratelimit:10m rate
            =5r/s;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # Logging Settings
        ##
        log_format main 'Event Source Information - Server Name: "
            $server_name" Server Protocol: "$server_protocol"
            Hostname: "$hostname" Host: "$host" '
    'Date and Timestamp Information - Local Time: "$time_local"
        ISO8601 Time: "$time_iso8601" '
    'Username Information - Basic Authentication User:
    "$remote_user" '
    'Source Address Information - Source Address:Port: "
        $remote_addr:$remote_port" '
    'Destination Address Information - Desination Address:Port "
        $server_addr:$server_port " '
    'Request Information - Request: "$request" HTTP Response Status
        : "$status" Referer: "$http_referer" Content Type:
    "$content_type" Body Bytes Sent: "$body_bytes_sent" User Agent:
         "$http_user_agent" ';

        access_log /var/log/nginx/access.log main;
        error_log /var/log/nginx/error.log info;

        ##
        # Gzip Settings
        ##
        gzip off;

  ##
  # Basic hardening
  ##
  # Disables the showing of nginx version
  server_tokens off;

  # Limit buffer sizes for clients
  client_max_body_size 100K;
  large_client_header_buffers 2 1k;
  client_body_buffer_size 1k;
  client_header_buffer_size 1k;

  # Enforce timeouts
  client_body_timeout 10;
  client_header_timeout 10;
  send_timeout 10;

        ##
        # Virtual Host Configs
        ##
        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}

######## /etc/nging/sites-enabled/file_server.conf ########
# Managed by Ansible

server {
    listen 443 ssl http2;
    server_name boot.group2.com;

    include /etc/nginx/common.conf;
    include /etc/nginx/ssl.conf;

    # Hide hidden files
    location ~ /\. {
      deny all;
      return 404;
    }

    location / {
      # Only respond to GET and HEAD requests
      limit_except GET HEAD { deny all; }

      # Only allow requests from one subnet
      allow   10.1.1.0/24;
      deny    all;

      # Limits the connections per ip
      limit_conn limitperip 10;
      limit_req zone=ratelimit burst=10 nodelay;

      autoindex on;
      root /var/www;
    }
}

######## /etc/nging/sites-enabled/http_to_https.conf ########
# Managed by Ansible

# Redirect everything to https
server {
    listen 80 ;
    server_name boot.group2.com;
    return 301 https://$host$request_uri;
}

######## /etc/nging/ssl.conf ########
# Managed by Ansible

ssl_certificate           /etc/ssl/certs/nginx-selfsigned.crt;
ssl_certificate_key       nginx-selfsigned.key;

ssl_protocols               TLSv1.2 TLSv1.3;
ssl_ciphers                 HIGH:!EXP:!NULL:!ADH:!LOW:!SSLv2:!SSLv3
    :!MD5:!RC4;
ssl_ecdh_curve              secp384r1;
ssl_prefer_server_ciphers   on;
```

```
ssl_session_timeout         10m;
ssl_session_cache           shared:SSL:10m;
ssl_session_tickets         off;
ssl_stapling                on;
ssl_stapling_verify         on;
ssl_dhparam                 /etc/nginx/dhparams.pem;

######## /etc/nging/common.conf ########
# Managed by Ansible

add_header Strict-Transport-Security    "Strict-Transport-Security:
     max- age=31536000; includeSubDomains; preload";
add_header X-Frame-Options              "SAMEORIGIN" always;
add_header X-Content-Type-Options       "nosniff" always;
add_header X-XSS-Protection             "1; mode=block";
add_header Referrer-Policy              "no-referrer";
add_header Content-Security-Policy      "default-src 'self';"
    always;
```

Listing 6: Example NGINX configuration files with example values to illustrate an actual configuration.

# O Table-driven Test Example

```
func TestAddClient(t *testing.T) {
        setup()

        type args struct {
                client structs.Client
        }
        tests := []struct {
                name          string
                args          args
                expectedError error
        }{
                {
                        name: "Add client without UUID",
                        args: args{
                                client: structs.Client{
                                        Status: []structs.Status{},
                                },
                        },
                        expectedError: nil,
                },
                {
                        name: "Add client with UUID",
                        args: args{
                                client: structs.Client{
                                        UUID:   "3",
                                        Status: []structs.Status{},
                                },
                        },
                        expectedError: ErrClientIDGiven,
                },
        }
        for _, tt := range tests {
                t.Run(tt.name, func(t *testing.T) {
                        if _, err := AddClient(tt.args.client);
                !errors.Is(err, tt.expectedError) {
                                t.Errorf("Addclient() error = %v,
                                    expectedError %v", err, tt.
                                    expectedError)
                        }
                })

        }
}
```

Listing 7: An example test that uses table-driven testing.

# P System Administrator Guide

# System administrator guide

This document describes how to deploy the automatic provisioning system. It describes the required steps for a simple installation on one or more virtual machines or bare metal servers. These systems should run Red Hat Enterprise Linux 9.

## Prerequisites

There are some prerequisites that need to be in place before the automatic provisioning system. These are:

- One or more virtual machines or bare metal servers running Red Hat Enterprise Linux 9.
    - For a single host installation a minimum of 4 GB of RAM and two cores is recommended.
    - These machines should be configured in accordance with the company's policies.
    - With access to a package repository containing the needed packages.
- A proficiently secured network.
- A machine to run the Ansible playbook from.
- Access to the git repository containing the Ansible playbook and roles.
- A working instance of Ansible Automation Platform.

## Installation

This section describes how to install the system on one or more hosts.

1. Clone the http_boot_server repository to the machine that will run the Ansible playbook.
2. Install the required roles and collections by running the following commands:

```
ansible-galaxy install -r roles/requirements.yml
ansible-galaxy install -r collections/requirements.yml
```

3. Create a file called `inventory` . This will be your inventory file, containing the hosts that the provisioning system should be installed on. A single or multiple servers can independently be assigned to the different tags in the inventory file. It is also possible to exclude services that are already being fulfilled outside the provisioning system.
    i. Use the group tag `[http_boot_server]` to specify the hosts that should act as the http file server.
    ii. Use the group tag `[dhcp_server]` to specify the hosts that should act as the DHCP server.
    iii. Use the group tag `[dns_server]` to specify the hosts that should act as the DNS server.
    iv. Use the group tag `[orchestrate]` to specify the hosts that should act as the orchestrate server (there should only be one instance).
4. Configure the system through Ansible variables. These should be set in `group_vars` folder. The name of the file corresponds to the group tag in the inventory file. For what variables are available, see

the `README.md` file in `http_boot_server` repository.

5. Move the TLS certificates to the host. Both the NGINX http server and the orchestration service should be run using TLS. The certificates and keys should be placed in the folders specified in the Ansible variables. Self-signed certificates can for example be created using the following command. However, the certificates should be customized to the company's needs:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/selfs
```

6. When the system is configured, run the playbook by running the following command from the `http_boot_server` folder:

```
ansible-playbook -i inventory --ask-become-pass main.yml
```

When the playbook has finished running, the system should be ready to use. To verify that the system is working, try connecting a computer to the network and see if the DHCP and DNS services work. After that, using a browser, verify that the orchestration service is available and that the NGINX http file server is working.

# Add install configuration to the system

This section describes how to add installation configurations to the boot server.

## Prerequisites

- The desired operative system image file
- A kickstart file
- A job template in Ansible Automation Platform

1. In the web root folder of NGINX, create a folder with the name of the configuration.
2. Add the required pre- and post-scripts to the kickstart file in order to communicate with the orchestration service. Change the `ORCHESTRATOR_IP` with the ip of the one of your orchestrator deployment. NB: This is not a complete kickstart file, only the relevant parts are shown.

```
%pre --log=PRE_LOG_LOCATION
# Download the live image to file to avoid tls-errors
curl --insecure -o /tmp/liveimg.tar.gz https://FILE_SERVER_IP/CONF_LOCATION/liveimg.tar

# Register client in orchestrator and store the ID
curl -ks -X POST -H "Content-Type: application/json" -d '{
"script-location": "http://FILE_SERVER_IP/CONF_LOCATION/"
}' https://ORCHESTRATOR_IP/api/v1/client/?only-id=true > /tmp/id.txt


# Add the ID as a variable
id=$(cat /tmp/id.txt)

# Change status to installing_os in orchestrator
curl -kX POST -H "Content-Type: application/json" -d '{
```

```
"client-id": "'"$id"'",
"state": "installing_os"
}' https://ORCHESTRATOR_IP/api/v1/status/

%end
```

```
# POST section run in no-chroot environment
%post --nochroot --log=POST_LOG_LOCATION

# Copy id to chroot environment
cp /tmp/id.txt /mnt/sysimage/etc/id.txt

%end
```

```
# POST section run in chroot environment
%post --log=POST_LOG_LOCATION

# Read the response from the file created in the %pre section
id=$(cat /tmp/id.txt)

# Change status to installed_os in orchestrator
curl -kX POST -H "Content-Type: application/json" -d '{
"client-id": "'"$id"'",
"state": "installed_os"
}' https://ORCHESTRATOR_IP/api/v1/status/

# Create the status_rebooted.sh script
cat > /etc/status_rebooted.sh << 'EOL'
#!/bin/bash
id=$(cat /etc/id.txt)

curl -kX POST -H "Content-Type: application/json" -d '{
"client-id": "'"$id"'",
"state": "rebooted"
}' https://ORCHESTRATOR_IP/api/v1/status/

curl -kX POST -H "Content-Type: application/json" -d '{
"client-id": "'"$id"'",
"state": "ansible_ready"
}' https://ORCHESTRATOR_IP/api/v1/status/

EOL

# Make the script executable
chmod +x /etc/status_rebooted.sh

# Create a systemd service to run status_rebooted.sh at the first boot
cat > /etc/systemd/system/status-rebooted.service << EOL
[Unit]
Description=Status Rebooted Script
After=network-online.target
Wants=network-online.target

[Service]
Type=oneshot
ExecStart=/etc/status_rebooted.sh
```

```
ExecStartPost=/bin/systemctl disable --now status-rebooted.service
RemainAfterExit=yes

[Install]
WantedBy=default.target
EOL

# Enable the systemd service
systemctl enable status-rebooted.service
%end
```

3. Move the kickstart file (must be called `kickstart.cfg` ) and operative system image to the folder created in step 1.
4. In the folder, create a file called `config.yml` and the desired configuration. An example of a configuration file:

```
job_templates:
  - template_id: 8
    continue_on_failure: false
  - template_id: 10
    continue_on_failure: true
```

5. To be able to boot into the operative system images, it must be unpacked. This can be done by running the following command:

```
bsdtar -xf <image_name>.iso -C <path_to_folder>
```

6. Add the configuration to the `grub_menu_entries` variable in the `group_vars` file for the http boot server, see the `README.md` file in `http_boot_server` repository for the configuration options.
7. Run the `http_boot_server` playbook again to ensure the proper file permissions are set and to update the grub configuration:

```
ansible-playbook -i inventory --ask-become-pass main.yml
```

# Q User Guide

# User guide

This guide describes how a user can perform a network installation using the automatic provisioning system.

## Prerequisites

Some prerequisites are needed to perform the installation.

- A USB stick with the CA certificate
- Admin access to the UEFI of the client
- Secure boot must be turned off on the computer

## Dell UEFI

This is the steps that need to be done on a Dell client for it to HTTP boot.

1. Turn on `Advanced Setup`
2. Make sure the `UEFI Network Stack` is enabled
3. Make sure `HTTP(S) Boot` is enabled
4. If the DHCP server is enabled with the http boot option, leave the `HTTP(S) Boot Modes` to `Auto Mode`, else set it to `Manual Mode`.
    i. If in `Manual Mode`, specify the boot url. This includes the schema and the path to the boot file.
5. Upload the CA certificate to the UEFI (without this step, the booting will fail) (the certificate must have the file ending `.pem` or `.PEM`). See Dell's documentation
    i. The USB drive with the certificate must be plugged in to the computer during the duration of the UEFI booting.

If errors occurs, see this Dell page to interpret the error message here.

## Installation

1. Boot the laptop with HTTP boot.
2. When the boot menu appears, select the desired configuration option.
3. To monitor the provisioning process, open a browser and use the URL for the orchestration service:
    i. First, the client ID must be fetched from the `/api/v1/client/` endpoint. Example output below:

```
{
  "cc0b3dba-29ef-4a39-a5c6-3ae883538af4": {
    "id": "cc0b3dba-29ef-4a39-a5c6-3ae883538af4",
    "script-location": "https://boot1.group2.com/rhel9/",
    "status": [
        {
          "state": "registered",
          "ip-address": "10.12.169.112",
          "timestamp": "2023-05-03T13:36:11.182871336+02:00",
          "message": ""
        },
        {
          "state": "installing_os",
          "ip-address": "10.12.169.112",
          "timestamp": "2023-05-03T13:36:11.231689438+02:00",
          "message": ""
        }
    ],
  "active": true
  }
}
```

ii. Then, go to the `/api/v1/status/<client_id>` endpoint to see the current status of the provisioning process. Example output below:

```
{
   "state": "installing_os",
   "ip-address": "10.12.169.112",
   "timestamp": "2023-05-03T13:36:11.231689438+02:00",
   "message": ""
}
```

4. When the provisioning process is finished, the status will be `configured` .

If errors occur during the installation process, check the status endpoint to see the error message. For more extensive logs, check the client or provisioning system logs

## R Pipeline for the Orchestration Service

```
stages:
  - test
  - sec
  - build
test:
  image: golang:1.18.9
  stage: test
  script:
    - go mod download
    - go get orchestrator/internal/clients
    - go get orchestrator/internal/config
    - go get orchestrator/internal/logging
    - go get github.com/boumenot/gocover-cobertura
    - go test ./... -coverprofile=coverage.txt -covermode atomic
    - go get gotest.tools/gotestsum@latest
    - go run gotest.tools/gotestsum --junitfile report.xml --format
        testname
    - go run github.com/boumenot/gocover-cobertura < coverage.txt >
        coverage.xml
    - go tool cover -func=coverage.txt
  artifacts:
    when: always
    reports:
      junit: report.xml
      coverage_report:
        coverage_format: cobertura
        path: coverage.xml
  coverage: '/^total:\s+\(statements\)\s+(\d+\.\d+)%/'
  rules:
    - if:
      exists:

sast:
  stage: sec
include:
  - template: Security/SAST.gitlab-ci.yml
  - template: Security/Dependency-Scanning.gitlab-ci.yml
  - template: Security/Secret-Detection.gitlab-ci.yml

build:
  stage: build
  image: golang:1.18.9
  script:
    - cd cmd/
    - go build -a -ldflags '-extldflags "-static"' -o orchestrator
  artifacts:
    paths:
      - cmd/orchestrator
  rules:
    - if: '$CI_COMMIT_BRANCH == "master"'
    - if: '$CI_COMMIT_MESSAGE =~ /Build executables/i'
```

Listing 8: The pipeline configuration used in the orchestration service project.

# S NGINX CIS Benchmark Report

# Appendix: Summary Table

| CIS Benchmark Recommendation | | Set Correctly | |
|---|---|---|---|
| | | Yes | No |
| **1** | **Initial Setup** | | |
| **1.1** | **Installation** | | |
| 1.1.1 | Ensure NGINX is installed (Automated) | ☒ | □ |
| 1.1.2 | Ensure NGINX is installed from source (Manual) | □ | ☒ |
| **1.2** | **Configure Software Updates** | | |
| 1.2.1 | Ensure package manager repositories are properly configured (Manual) | ☒ | □ |
| 1.2.2 | Ensure the latest software package is installed (Manual) | ☒ | □ |
| **2** | **Basic Configuration** | | |
| **2.1** | **Minimize NGINX Modules** | | |
| 2.1.1 | Ensure only required modules are installed (Manual) | □ | ☒ |
| 2.1.2 | Ensure HTTP WebDAV module is not installed (Automated) | □ | ☒ |
| 2.1.3 | Ensure modules with gzip functionality are disabled (Automated) | □ | ☒ |
| 2.1.4 | Ensure the autoindex module is disabled (Automated) | □ | ☒ |
| **2.2** | **Account Security** | | |
| 2.2.1 | Ensure that NGINX is run using a non-privileged, dedicated service account (Automated) | ☒ | □ |
| 2.2.2 | Ensure the NGINX service account is locked (Automated) | ☒ | □ |
| 2.2.3 | Ensure the NGINX service account has an invalid shell (Automated) | ☒ | □ |

| CIS Benchmark Recommendation | | Set Correctly | |
|---|---|---|---|
| | | Yes | No |
| **2.3** | **Permissions and Ownership** | | |
| 2.3.1 | Ensure NGINX directories and files are owned by root (Automated) | ☒ | □ |
| 2.3.2 | Ensure access to NGINX directories and files is restricted (Automated) | ☒ | □ |
| 2.3.3 | Ensure the NGINX process ID (PID) file is secured (Automated) | ☒ | □ |
| 2.3.4 | Ensure the core dump directory is secured (Manual) | ☒ | □ |
| **2.4** | **Network Configuration** | | |
| 2.4.1 | Ensure NGINX only listens for network connections on authorized ports (Manual) | ☒ | □ |
| 2.4.2 | Ensure requests for unknown host names are rejected (Automated) | □ | ☒ |
| 2.4.3 | Ensure keepalive_timeout is 10 seconds or less, but not 0 (Automated) | ☒ | □ |
| 2.4.4 | Ensure send_timeout is set to 10 seconds or less, but not 0 (Automated) | ☒ | □ |
| **2.5** | **Information Disclosure** | | |
| 2.5.1 | Ensure server_tokens directive is set to `off` (Automated) | ☒ | □ |
| 2.5.2 | Ensure default error and index.html pages do not reference NGINX (Automated) | □ | ☒ |
| 2.5.3 | Ensure hidden file serving is disabled (Manual) | ☒ | □ |
| 2.5.4 | Ensure the NGINX reverse proxy does not enable information disclosure (Automated) | □ | □ |
| **3** | **Logging** | | |

N/A

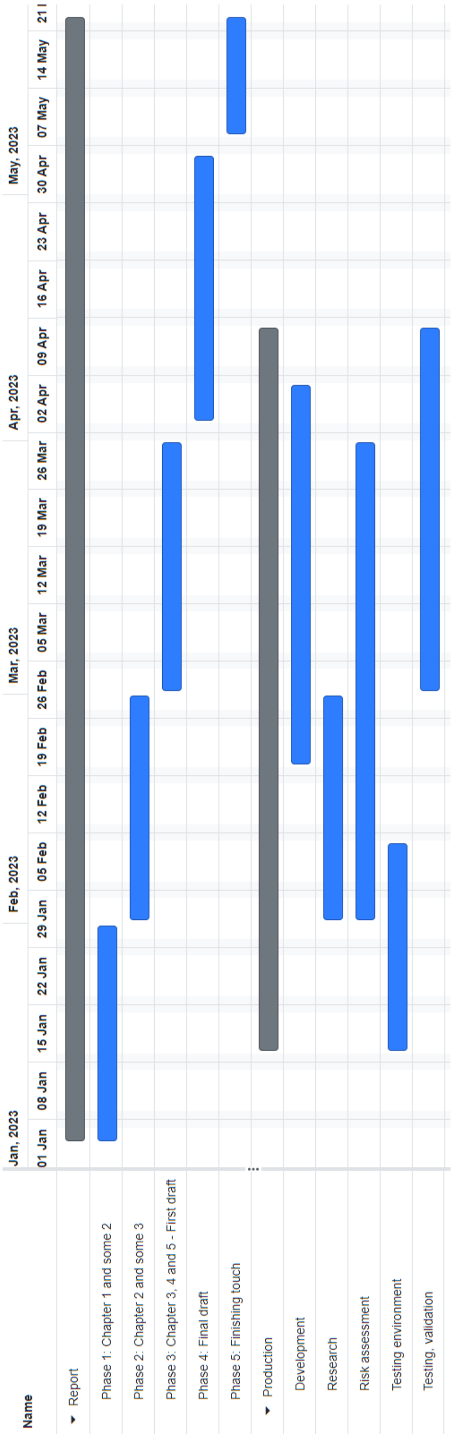
| CIS Benchmark Recommendation | | Set Correctly | | |
|---|---|---|---|---|
| | | Yes | No | |
| 3.1 | Ensure detailed logging is enabled (Manual) | ☒ | □ | |
| 3.2 | Ensure access logging is enabled (Manual) | ☒ | □ | |
| 3.3 | Ensure error logging is enabled and set to the info logging level (Automated) | ☒ | □ | |
| 3.4 | Ensure log files are rotated (Automated) | ☒ | □ | |
| 3.5 | Ensure error logs are sent to a remote syslog server (Manual) | ☒ | □ | |
| 3.6 | Ensure access logs are sent to a remote syslog server (Manual) | ☒ | □ | |
| 3.7 | Ensure proxies pass source IP information (Manual) | □ | □ | NA |
| **4** | **Encryption** | | | |
| **4.1** | **TLS / SSL Configuration** | | | |
| 4.1.1 | Ensure HTTP is redirected to HTTPS (Manual) | ☒ | □ | |
| 4.1.2 | Ensure a trusted certificate and trust chain is installed (Manual) | □ | □ | NA |
| 4.1.3 | Ensure private key permissions are restricted (Automated) | □ | □ | NA |
| 4.1.4 | Ensure only modern TLS protocols are used (Automated) | ☒ | □ | |
| 4.1.5 | Disable weak ciphers (Manual) | ☒ | □ | |
| 4.1.6 | Ensure custom Diffie-Hellman parameters are used (Automated) | ☒ | □ | |
| 4.1.7 | Ensure Online Certificate Status Protocol (OCSP) stapling is enabled (Automated) | ☒ | □ | |
| 4.1.8 | Ensure HTTP Strict Transport Security (HSTS) is enabled (Automated) | ☒ | □ | |

| CIS Benchmark Recommendation | | Set Correctly | | |
|---|---|---|---|---|
| | | Yes | No | |
| 4.1.9 | Ensure upstream server traffic is authenticated with a client certificate (Automated) | □ | □ | NA |
| 4.1.10 | Ensure the upstream traffic server certificate is trusted (Manual) | □ | □ | NA |
| 4.1.11 | Ensure your domain is preloaded (Manual) | ☒ | □ | |
| 4.1.12 | Ensure session resumption is disabled to enable perfect forward security (Automated) | ☒ | □ | |
| 4.1.13 | Ensure HTTP/2.0 is used (Automated) | ☒ | □ | |
| 4.1.14 | Ensure only Perfect Forward Secrecy Ciphers are Leveraged (Manual) | □ | ☒ | |
| **5** | **Request Filtering and Restrictions** | | | |
| **5.1** | **Access Control** | | | |
| 5.1.1 | Ensure allow and deny filters limit access to specific IP addresses (Manual) | ☒ | □ | |
| 5.1.2 | Ensure only approved HTTP methods are allowed (Manual) | ☒ | □ | |
| **5.2** | **Request Limits** | | | |
| 5.2.1 | Ensure timeout values for reading the client header and body are set correctly (Automated) | ☒ | □ | |
| 5.2.2 | Ensure the maximum request body size is set correctly (Automated) | ☒ | □ | |
| 5.2.3 | Ensure the maximum buffer size for URIs is defined (Automated) | ☒ | □ | |
| 5.2.4 | Ensure the number of connections per IP address is limited (Manual) | ☒ | □ | |
| 5.2.5 | Ensure rate limits by IP address are set (Manual) | ☒ | □ | |

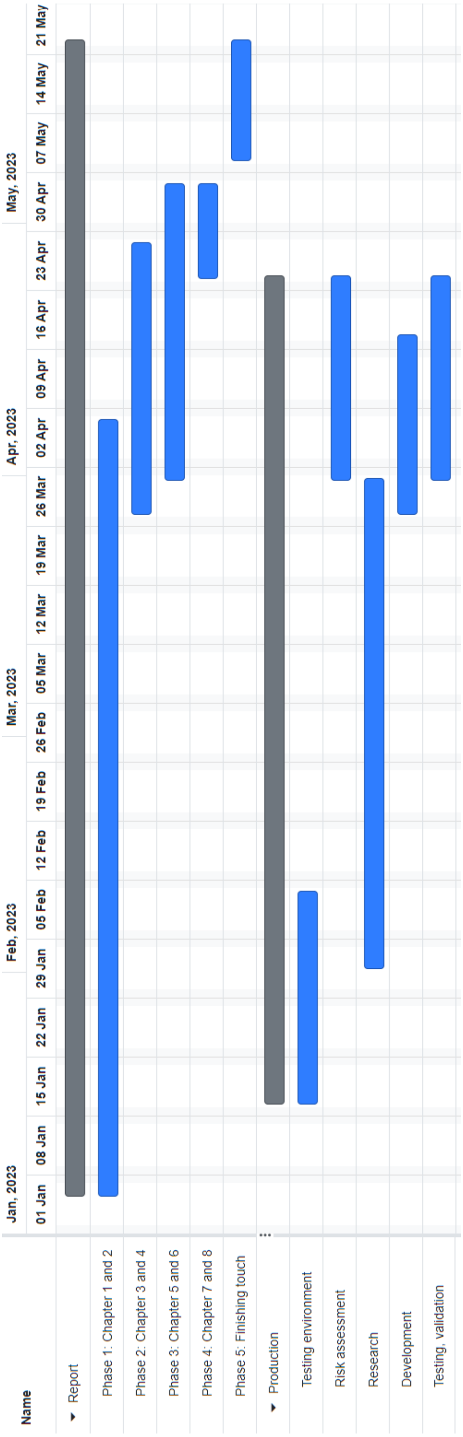
| CIS Benchmark Recommendation | | Set Correctly | |
|---|---|---|---|
| | | Yes | No |
| **5.3** | **Browser Security** | | |
| 5.3.1 | Ensure X-Frame-Options header is configured and enabled (Automated) | ☑ | □ |
| 5.3.2 | Ensure X-Content-Type-Options header is configured and enabled (Automated) | ☒ | □ |
| 5.3.3 | Ensure that Content Security Policy (CSP) is enabled and configured properly (Manual) | ☒ | □ |
| 5.3.4 | Ensure the Referrer Policy is enabled and configured properly (Manual) | ☒ | □ |
| **6** | **Mandatory Access Control** | | |

# T Gantt Diagrams

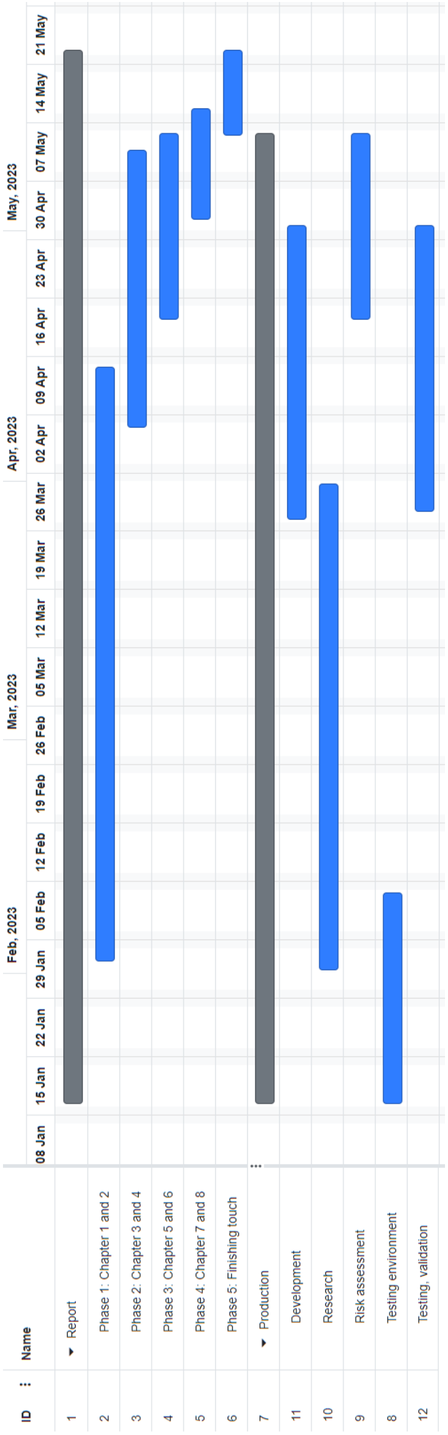
Gantt diagram from project plan:



Revised Gantt diagram:



Final Gantt diagram:

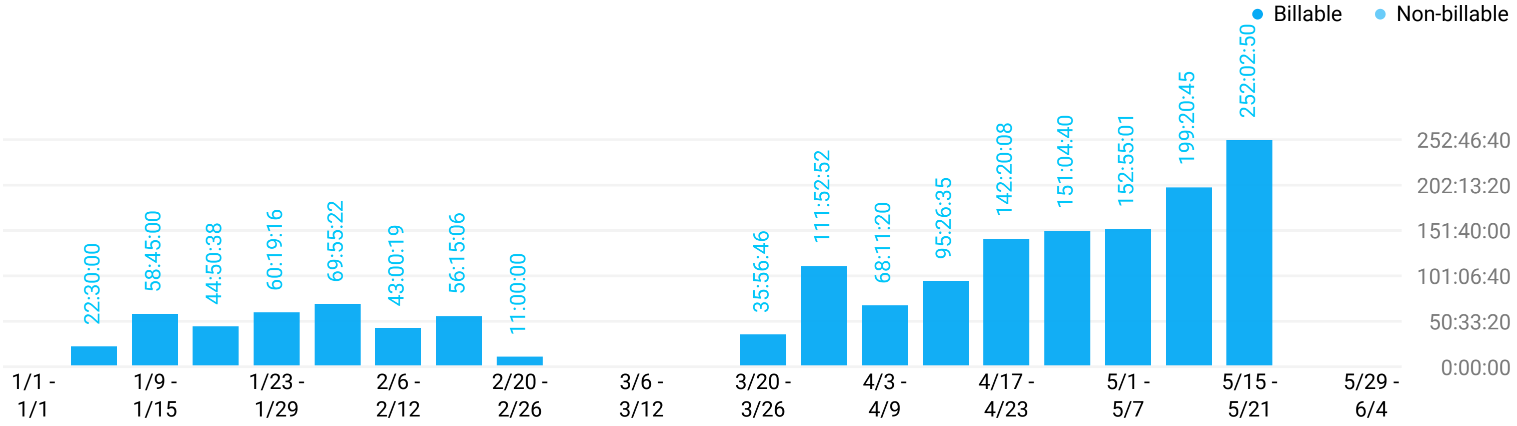


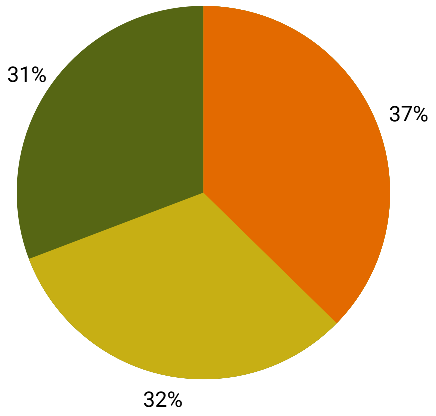
# U Time Tables

# Summary Report

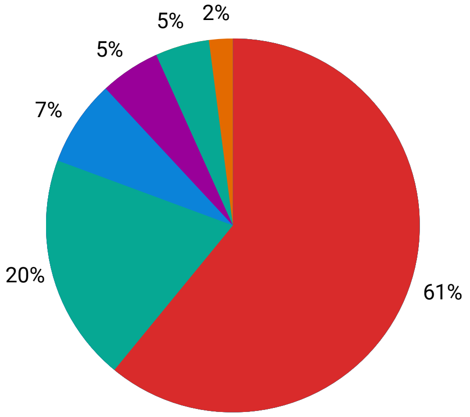
01/01/2023 – 05/31/2023

| TOTAL HOURS | BILLABLE HOURS |
|---|---|
| 1575:46:38 | 1575:46:38 |





| USER | DURATION |
|---|---|
| MA Mathiws | 587:56:30 |
| KA Karishms | 502:05:24 |
| VE Vegarfae | 485:44:44 |



| TIME ENTRY | DURATION |
|---|---|
| ● Documentation | 960:31:07 |
| ● Coding | 310:36:13 |
| ● Meeting | 117:25:22 |
| ● Project planning | 80:27:32 |
| ● Research | 73:22:48 |
| ● Infrastructure | 33:23:36 |

| USER - TIME ENTRY | DURATION | AMOUNT | PERCENTAGE |
|---|---|---|---|
| KA Karishms | 502:05:24 | 0.00 USD | 31.86% |
| Coding | 42:29:52 | 0.00 USD | 2.7% |
| Documentation | 366:04:16 | 0.00 USD | 23.23% |
| Infrastructure | 6:30:00 | 0.00 USD | 0.41% |
| Meeting | 41:34:01 | 0.00 USD | 2.64% |
| Project planning | 23:49:47 | 0.00 USD | 1.51% |
| Research | 21:37:28 | 0.00 USD | 1.37% |
| MA Mathiws | 587:56:30 | 0.00 USD | 37.31% |
| Coding | 148:21:21 | 0.00 USD | 9.41% |
| Documentation | 316:56:42 | 0.00 USD | 20.11% |
| Infrastructure | 22:23:36 | 0.00 USD | 1.42% |
| Meeting | 32:39:22 | 0.00 USD | 2.07% |
| Project planning | 37:50:09 | 0.00 USD | 2.4% |
| Research | 29:45:20 | 0.00 USD | 1.89% |
| VE Vegarfae | 485:44:44 | 0.00 USD | 30.83% |
| Coding | 119:45:00 | 0.00 USD | 7.6% |
| Documentation | 277:30:09 | 0.00 USD | 17.61% |

| USER - TIME ENTRY | DURATION | AMOUNT | PERCENTAGE |
|---|---|---|---|
| Infrastructure | 4:30:00 | 0.00 USD | 0.29% |
| Meeting | 43:11:59 | 0.00 USD | 2.74% |
| Project planning | 18:47:36 | 0.00 USD | 1.19% |
| Research | 22:00:00 | 0.00 USD | 1.4% |

# V Kickstart Example Configurations

```
#version=RHEL9
# Use graphical install
graphical
liveimg --url file:///tmp/liveimg.tar.gz

%addon com_redhat_kdump --enable --reserve-mb='auto'
%end

# Keyboard layouts
keyboard --xlayouts='no'
# System language
lang en_GB.UTF-8

# Network information
network  --bootproto=dhcp --device=enX0 --ipv6=auto --activate

# Use CDROM installation media
cdrom

%packages
@^minimal-environment
jq
curl
%end

# Run the Setup Agent on first boot
firstboot --enable

# Partitioning
reqpart
ignoredisk --only-use=nvme0n1
zerombr
clearpart --all --initlabel
part /boot/efi --fstype=xfs --size=600
part /boot --fstype=xfs --size=1024
volgroup system --pesize=4096 pv.2
part pv.2 --fstype=lvmpv --ondisk=nvme0n1 --size=230000

# System timezone
timezone Europe/Oslo --utc

# Root password
rootpw --iscrypted iuerhgv7eynt89n4yhgob8iuepomrjhpg89hjroigphse0ø
    o9h5
user --groups=wheel --name=admin --password=
    durhvm9e78ypgvoiehprg98oseøgpesiuhoigespg8oisheøogp89 --
    iscrypted

# Reboot the client after install
reboot

%pre --log=/root/ks-pre.log
# Download the live image to file
curl --insecure -o /tmp/liveimg.tar.gz https://10.43.54.64/rhel9/
    liveimg.tar.gz

# Register client in orchestrator and store the ID
curl -ks -X POST -H "Content-Type: application/json" -d '{
                                    "script-location": "http
                                        ://10.43.54.64/rhel9/"
}' https://10.43.54.64:8080/api/v1/client/?only-id=true > /tmp/id.
    txt

id=$(cat /tmp/id.txt)

# Change status to installing_os in orchestrator
curl -kX POST -H "Content-Type: application/json" -d '{
```

```
                                "client-id": "'"$id"'",
                                "state": "installing_os"
}' https://10.43.54.64:8080/api/v1/status/

%end

%post --nochroot --log=/root/ks-post-nochroot.log
# Read the response from the temporary file created in the %pre
    section
id=$(cat /tmp/id.txt)

# Copy id to chroot environment
cp /tmp/id.txt /mnt/sysimage/etc/id.txt

# Change status to installed_os in orchestrator
curl -kX POST -H "Content-Type: application/json" -d '{
                                "client-id": "'"$id"'",
                                "state": "installed_os"
}' https://10.43.54.64:8080/api/v1/status/

%end

%post --log=/root/ks-post.log
# Create the status_rebooted.sh script
cat > /etc/status_rebooted.sh << 'EOL'
#!/bin/bash
id=$(cat /etc/id.txt)

curl -kX POST -H "Content-Type: application/json" -d '{
    "client-id": "'"$id"'",
    "state": "rebooted"
}' https://10.43.54.64:8080/api/v1/status/

curl -kX POST -H "Content-Type: application/json" -d '{
                                "client-id": "'"$id"'",
                                "state": "ansible_ready"
}' https://10.43.54.64:8080/api/v1/status/

EOL

# Make the script executable
chmod +x /etc/status_rebooted.sh

# Create a systemd service to run status_rebooted.sh at the first
    boot
cat > /etc/systemd/system/status-rebooted.service << EOL
[Unit]
Description=Status Rebooted Script
After=network-online.target
Wants=network-online.target

[Service]
Type=oneshot
ExecStart=/etc/status_rebooted.sh
ExecStartPost=/bin/systemctl disable --now status-rebooted.service
RemainAfterExit=yes

[Install]
WantedBy=default.target
EOL

# Enable the systemd service
systemctl enable status-rebooted.service
%end
```

Listing 9: Example Kickstart configuration file

NTNU

Norwegian University of
Science and Technology