Securing Company Infrastructure with Modern Automation Tools

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Cyber Security Engineering
September 2025
Resnais Evenn

Supervisors:
Ismayil Hasanov
Petri Sainio

UNIVERSITY OF TURKU
Department of Computing

RESNAIS EVENN: Securing Company Infrastructure with Modern Automation Tools

Master of Science (Tech) Thesis, 64 p.
Cyber Security Engineering
September 2025

This thesis presents the design and implementation of an automated infrastructure using open-source virtualization and Infrastructure as Code (IaC) tools to streamline deployment and management processes. The proposed environment leverages a Proxmox Virtual Environment (PVE) hypervisor for virtualization, with HashiCorp Packer employed to create standardized VM templates. Terraform is used in conjunction with Ansible to provision and configure these virtual machines automatically, ensuring consistency and reducing manual effort. Docker containers host key services, and the Traefik reverse proxy routes external traffic to these services, enabling efficient access management. Security and identity management are integrated into the infrastructure by incorporating Keycloak for centralized authentication and Vault for secure secret storage. The automated pipeline builds the infrastructure from the ground up ranging from base VM images to fully configured services using definition files and scripts, thereby minimizing configuration errors and improving reproducibility. The thesis details the architecture of this automated system, the IaC workflow for provisioning and configuration, and the deployment of critical services within the environment. Finally, it evaluates the solution's effectiveness in simplifying infrastructure setup and discusses the benefits, limitations, and potential improvements of the approach.

Keywords: Infrastructure as Code, Proxmox, Terraform, Ansible, Keycloak, Vault, DevOps, Cybersecurity

Contents

List of Figures

1 Introduction

This introductory chapter establishes the foundation of the thesis. It outlines the motivation for the work, formulates the research problem and objectives, and clarifies the scope and expected contributions. The purpose is to provide the reader with the necessary context for understanding why the project was undertaken, what it seeks to achieve, and how it fits within the broader field of infrastructure automation and cybersecurity.

1.1 Background and Motivation

Small and medium-sized companies (SMEs) often face challenges when setting up and maintaining their IT infrastructure. These challenges include limited personnel, limited time, and the increasing need for secure and efficient systems. At the same time, expectations for reliability, availability, and cybersecurity have grown significantly in recent years. Companies are expected to provide internal services such as identity management, asset tracking, and user authentication in a secure and maintainable way.

Manual configuration of these systems can be slow and error-prone. It often results in inconsistencies and makes long-term maintenance more difficult. To address this, many organisations have started adopting automation tools and open-source solutions that support Infrastructure-as-Code (IaC) principles. These tools make it possible to define infrastructure and services in code, allowing repeatable and

consistent deployment [1].

Modern tools such as Proxmox Virtual Environment (Proxmox VE) for virtualization, Terraform for infrastructure provisioning, Ansible for configuration management, and Docker for containerization have become widely used in both industry and research. When used together, these tools can help create a modular and secure infrastructure that is easier to deploy and maintain [2]. Additionally, tools like Traefik can be used as reverse proxies to improve service access and security by controlling routing, handling SSL certificates, and adding a layer of abstraction between users and backend systems [3].

The motivation for this thesis arises from my experience in a professional environment, during which I was assigned to build a basic infrastructure setup using these technologies. My goal was to learn how to use these tools together to create a working environment that is secure, modular, and suited to the needs of a small company. This included setting up key services such as Active Directory (AD), Snipe-IT, and Keycloak, and managing them under a Domain Name System (DNS) and reverse proxy. The experience also helped me understand how automation can support better security and reproducibility in infrastructure projects.

1.2 Research Problem and Objectives

The primary problem addressed in this thesis is how to build and manage a secure and automated IT infrastructure for a SME using modern, open-source tools. Many companies still rely on manual setup processes and ad-hoc configurations, which can introduce security vulnerabilities, increase maintenance overhead, and reduce system reliability. These problems become more significant when services such as identity management, internal asset tracking, and user access control are required.

There is a wide range of tools available for solving these problems, but combining them into a coherent, well-documented infrastructure that meets both security and

automation requirements is often difficult without prior experience. This challenge is particularly relevant in SMEs where IT resources are limited.

Using IaC helps reduce the risk of misconfiguration, supports auditing, and makes infrastructure more scalable and consistent across environments [4]. It also offers business-level advantages such as faster deployments, cost savings, and self-documentation [5].

The goal of this thesis is to design and implement a practical infrastructure setup using tools such as Proxmox VE, Terraform, Ansible, Docker, and Traefik. The setup includes essential services like AD, Snipe-IT, and Keycloak, and integrates them into a working system managed by code. The infrastructure must be reproducible, modular, and easy to maintain.

More specifically, the objectives of the thesis are:

- To learn and evaluate the use of Iac tools in a real-world environment.
- To deploy a functioning infrastructure composed of virtual machines (VM), containers, and core services.
- To automate the provisioning and configuration of these services using software like Terraform and Ansible.
- To improve the security of the infrastructure by managing access, encrypting secrets, and using reverse proxy techniques.
- To document the entire process clearly and make the setup reusable for similar use cases.

By completing these objectives, the thesis aims to show how SMEs can adopt open-source tools to create infrastructure that is both secure and maintainable, while reducing manual effort and configuration errors.

1.3 Scope and Contribution

The scope of this thesis focuses on demonstrating how a company infrastructure can be deployed using IaC principles. The main goal is not to provide a universal solution, but rather to share a practical example of how infrastructure can be automated, secured, and structured in a consistent way. Since every company has its own requirements, existing systems, and constraints, the choice of software and tools will naturally differ from one case to another.

This work presents the specific setup I developed using tools such as Proxmox VE, Terraform, Ansible, Docker, and Traefik. The setup includes services like AD for identity management, Snipe-IT for asset tracking, and Keycloak for identity federation. These tools were chosen based on the needs of the company and the goals of the deployment.

The key takeaway from this thesis is not to follow every step or replicate the same set of tools, but to understand the architecture and principles behind the setup. Concepts such as automation, modularity, reproducibility, and separation of concerns are widely applicable, regardless of the exact technologies used [6]. The idea of modular design and the flexibility to adapt components based on organisational context are central to Iac and help make infrastructure scalable and compliant [7].

The contribution of this work lies in its practical and educational value. It provides a real-world example of infrastructure deployment using modern automation tools, along with documentation that can help others adapt similar methods to their own environment. It also reflects on the decisions made during the process, the challenges faced, and what lessons can be drawn from the experience.

The following chapters progressively develop the ideas introduced here. The work begins by reviewing the theoretical and technical background that underpins infrastructure automation and DevOps principles. It then details the system environment and the architectural decisions that structured the deployment. From

there, the focus shifts to the implementation and security integration, followed by a discussion of the setup's evaluation and results. The final chapter reflects on these outcomes, highlighting opportunities for improvement and outlining directions for future enhancements.

As a final note, portions of this thesis were refined and rephrased with the assistance of AI tools to enhance clarity, ensure grammatical accuracy, and verify adherence to standard format.

2 Background and Literature Review

This chapter provides an overview of the fundamental concepts and technologies used to implement the automated infrastructure. It introduces the virtualization platform and tools for IaC, as well as supporting technologies for containerization, networking, and security services. By understanding these components, one can appreciate how they interconnect to form a cohesive automated environment. The sections that follow discuss Proxmox VE for virtualization, HashiCorp Packer for template creation, Terraform and Ansible for infrastructure provisioning and configuration, Docker for containerization, the Traefik proxy for traffic management, and Keycloak and Vault for identity and secrets management.

2.1 Overview of Infrastructure Automation and DevOps

As IT environments grow in complexity, the need for scalable, secure, and reproducible infrastructure has become increasingly important. Infrastructure automation and DevOps practices aim to address these challenges by promoting a culture of continuous improvement, consistent configuration, and tighter integration between development and operations. Rather than relying on manual, error-prone setup

processes, infrastructure is now often defined using code and deployed through automated workflows.

A core concept in this domain is Iac, where infrastructure elements such as VMs, networking, storage, and services are described through declarative or procedural code. This approach ensures that configurations are repeatable, auditable, and version-controlled. Tools like Terraform are used for provisioning resources, while tool like Ansible handles the configuration and deployment of services. As shown by Koneru in their evaluation of enterprise practices [8], this model significantly improves system reliability, reduces maintenance effort, and allows small teams to handle complex environments efficiently.

DevOps complements this by introducing a set of practices that promote collaboration between developers and system administrators, with a focus on automation, continuous delivery (CD), and feedback loops. Automating infrastructure makes it easier to test changes, recover from failures, and maintain consistent environments across development, staging, and production.

Security remains an essential consideration in this automated approach. Real-world infrastructure often requires access to sensitive information such as API tokens, credentials, and database secrets. Embedding such data in code poses significant risks, especially in production environments. To manage this, secret management systems like HashiCorp Vault are used to store and control access to sensitive data in a secure and centralized way. During my experience, Vault was deployed and integrated into the infrastructure to ensure that secrets were encrypted, logged, and only accessible based on strict access controls.

Figure 2.1 illustrates a simplified architecture of the automated infrastructure, showing how different components interact from Iac tools running on the developer's machine, to the services deployed on VMs via Docker within the Proxmox host.

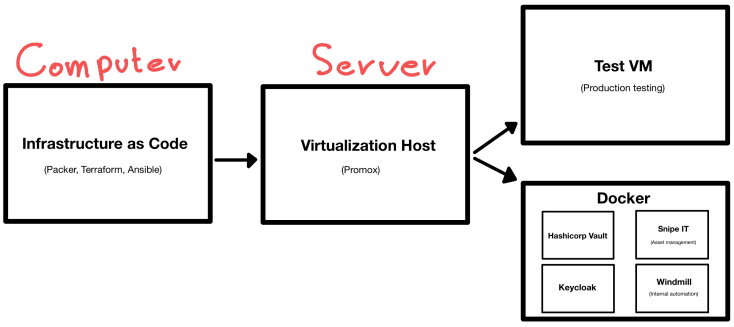

Figure 2.1: Simplified architecture of the automated infrastructure

Although such practices are often seen in large-scale organizations, they are increasingly relevant for SME. With limited IT personnel and budgets, SMEs benefit from systems that are reproducible, transparent, and easy to manage. Instead of relying on undocumented setups or manual intervention, automated workflows make systems easier to audit, scale, and maintain.

This thesis presents how these concepts were applied in a practical infrastructure setup. Rather than promoting a one-size-fits-all solution, the goal is to demonstrate key design principles that can be adapted based on specific organisational needs and constraints.

2.2 Key Technologies: Proxmox, Terraform, Ansible, Docker, Keycloak, and Vault

The infrastructure discussed in this work relies on several modern, open-source technologies that together provide virtualization, configuration management, service or-

chestration, identity management, and secure secrets handling.

Proxmox Virtual Environment (PVE)

Proxmox Virtual Environment (PVE) is an open-source virtualization platform for managing VMs and containers. It combines the Kernel-based VM (KVM) hypervisor with LXC container management into a single solution, providing a web-based interface for orchestrating compute, storage, and network resources. PVE enables the creation and snapshotting of VMs, virtual networks, and storage volumes, which makes it well suited for building and testing complex lab environments. Key features of Proxmox include support for high availability clusters and centralized management of multiple host nodes. In this project, Proxmox serves as the foundation on which all VMs are hosted and networked. By using Proxmox, the infrastructure gains a stable and feature-rich virtualization layer that can be programmatically controlled via APIs and command-line tools, facilitating its integration into the Iac workflow. The Proxmox environment used is visualized in Figure 2.2.

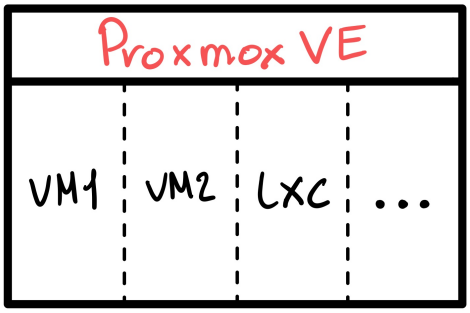

Figure 2.2: Proxmox VE Managing VMs and Containers

HashiCorp Packer

HashiCorp Packer is an open-source tool for creating identical machine images for multiple platforms from a single source configuration. In the context of this project, Packer is used to automate the building of VM templates for Proxmox. Instead of manually installing an operating system and base software on each VM, a Packer template (configuration file) defines the steps to create a standardized image. This image may include the operating system, updates, and any prerequisite software or configurations needed for the infrastructure's services. Packer then runs this build process in an automated fashion, producing a reusable VM template. By using Packer, the project ensures that all VMs start from a consistent baseline image, which improves reliability and reduces the setup time for new VMs. The Packer-built template is stored in Proxmox and later used by Terraform to instantiate multiple VMs with identical configurations.

Terraform and Ansible

Terraform and Ansible are Iac tools used in tandem. Terraform handles the provisioning of VMs and system resources, while Ansible is responsible for their configuration. This separation of concerns increases clarity and reusability of the automation logic. According to Byzov (2025), combining both tools provides a structured yet flexible approach to automating deployment workflows across hybrid environments [9]. Pathak (2024) also highlights the advantages of integrating Terraform and Ansible for infrastructure repeatability and system reliability [10].

The logical separation between provisioning and configuration phases is illustrated in Figure 2.3.

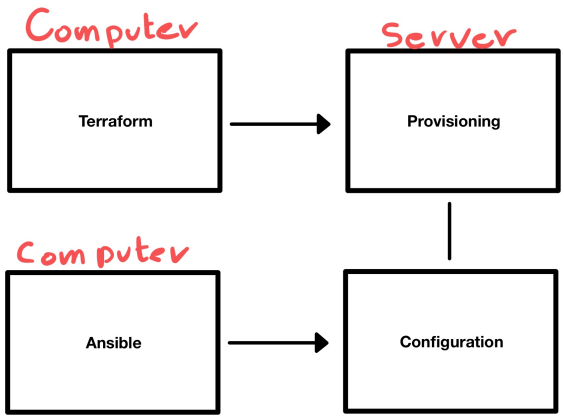

Figure 2.3: Separation Between Provisioning (Terraform) and Configuration (Ansible)

Docker and Docker Compose

Docker is a widely adopted platform that enables the packaging of applications and their dependencies into lightweight, portable containers. These containers can run consistently across different environments, reducing compatibility issues between development, testing, and production. In this project, Docker is used to deploy various services in a modular way.

Docker Compose complements Docker by allowing the definition and management of multi-container applications using a single YAML configuration file. This file specifies the services, networks, and volumes involved in an application stack, making deployments more maintainable and reproducible.

In the implemented infrastructure, Docker Compose is used to manage services such as Snipe-IT (asset management), Keycloak (identity and access management), and HashiCorp Vault (secrets management). Running them inside containers enables better isolation, faster redeployment, and simplified dependency management. Since all containers are defined and version-controlled, the infrastructure can be redeployed in a consistent state whenever needed, aligning with Iac principles.

The containers are hosted on a Debian-based VM provisioned on Proxmox. The use of a single host for multiple services allows efficient use of resources and centralized logging, while keeping services logically separated. This Docker deployment layout is shown in Figure 2.4.

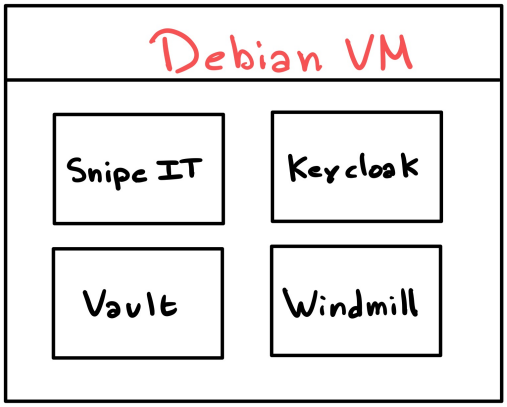

Figure 2.4: Service Deployment Inside Docker on a Debian VM

Keycloak

Keycloak is an open-source identity and access management (IAM) system developed by Red Hat. It provides features such as user federation, social login, two-factor authentication, and centralized authorization across applications. Keycloak also supports integration with existing user directories through LDAP or AD, as well as modern authentication protocols like OpenID Connect and SAML 2.0.

In this infrastructure, Keycloak plays a central role in unifying identity management across internal services. Instead of each service managing its own authentication mechanism, they delegate authentication to Keycloak. This allows consistent user policies, centralized credential management, and easier onboarding or deactivation of users.

The setup uses Docker to run Keycloak as a containerized service. Configuration parameters, such as realms, roles, and client applications, are defined using environ-

ment variables or initialization scripts to support automation. The service is made accessible through the Traefik reverse proxy and assigned a custom DNS entry to simplify access and improve security using HTTPS.

Keycloak was also chosen for its extendibility and support for role-based access control (RBAC). This allows fine-grained permissions to be assigned across services depending on user roles. This is particularly beneficial for small teams needing strong but manageable access control without building a custom solution from scratch.

HashiCorp Vault

Vault is a tool for secrets management, encryption, and access auditing. It is deployed as a container inside the Docker environment hosted on Proxmox. Vault enables the infrastructure to securely store sensitive values such as API keys, passwords, and tokens. Raghu (2025) presents Vault as a reliable solution for enterprise-level credential management and dynamic secrets handling [11]. This containerized Vault deployment helps enforce strict security boundaries between services while maintaining centralized access control.

2.3 Security Practices in Modern Infrastructure Design

Security is a foundational component of any IT infrastructure, especially in environments where automation, remote access, and service orchestration are involved. In modern DevOps workflows, security should not be treated as a separate layer but instead as an integral part of the infrastructure lifecycle. This section outlines the security principles and practices that were applied throughout the infrastructure implementation, based on proven methodologies and academic literature.

Principle of Least Privilege

The infrastructure applies the principle of least privilege (PoLP), ensuring that users, services, and systems have only the minimum access required to perform their roles. This limits the potential attack surface and minimizes the risk in case of compromise. For instance, the Ansible automation system uses service accounts with restricted roles, and containers are only permitted access to specific directories and ports needed for their operation.

Secure Secrets Management with Vault

Sensitive data such as API keys, database passwords, and administrative credentials must not be stored in plain text or exposed through configuration files. To address this, two mechanisms were used:

- **Ansible Vault** for encrypting provisioning variables and deployment secrets.
- **HashiCorp Vault**, deployed in a Docker container on a Proxmox-hosted VM, for runtime secret access control.

Vault offers dynamic secrets, role-based access control, and audit logging. Its integration improves both the confidentiality and integrity of sensitive information, particularly in a multi-service environment [11].

Reverse Proxy and TLS Encryption

To protect all HTTP-based services, a Traefik reverse proxy handles external requests and routes them securely to backend services such as Snipe-IT, Keycloak, and Vault. Traefik integrates with Let's Encrypt to automatically generate and renew TLS certificates. This ensures encrypted communication and avoids common pitfalls like expired certificates [12].

Container Isolation and Network Segmentation

Docker networks were used to enforce isolation between services. For example, Vault and Keycloak are deployed on separate internal networks that limit external access. Containers that need to communicate (e.g., frontend and backend of Snipe-IT) are explicitly placed on shared bridges.

The segmentation logic and routing architecture used to enforce these constraints is illustrated in Figure 2.5.

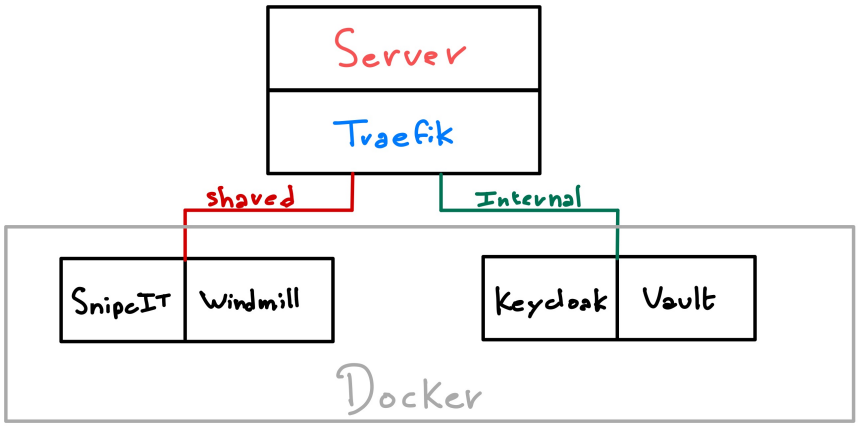

Figure 2.5: Traefik Reverse Proxy Setup for Shared and Internal Services

This segmentation reduces the chance of lateral movement in the event of a service compromise and improves container-level security posture.

Backup and Redundancy Strategies

Maintaining system availability requires robust backup mechanisms. Proxmox supports scheduled VM snapshots, and encrypted backups are planned using the Prox-

mox Backup Server. These backups allow for recovery in case of misconfigurations, system failures, or cyberattacks. Future improvements could include automatic offsite replication to further enhance resilience.

Infrastructure Documentation and Auditability

A secure infrastructure is not just about technology it also depends on process and traceability. All code used to deploy and configure the system is version-controlled. Changes are documented in Outline to ensure auditability, reproducibility, and better knowledge sharing across the team. Following DevSecOps principles, security was embedded into every layer of development and deployment [13].

Overall, this infrastructure combines well-established DevOps practices with embedded security principles to deliver a system that is both resilient and maintainable.

2.4 Related Work and Best Practices

The design and automation of IT infrastructure have evolved significantly in recent years, shaped by the rise of cloud computing, DevOps methodologies, and open-source tooling. This section explores how current academic and industry research aligns with the practices applied in this work. The aim is to contextualize the infrastructure setup presented in this thesis within the broader landscape of IaC, security-aware automation, and reproducible system design.

Infrastructure as Code in Practice

IaC has become a standard practice in system administration and cloud operations. It allows engineers to define computing resources, networks, and dependencies in version-controlled configuration files. According to Vanam (2025), IaC improves deployment speed, consistency, and resilience by reducing human intervention and

enabling environment replication [14]. Their systematic review highlights how organizations using tools like Terraform and Ansible report fewer misconfigurations and faster onboarding of new team members.

These findings directly support the goals of this thesis. The use of Terraform and Ansible throughout the infrastructure reflects these best practices, enabling infrastructure reproducibility across environments. Instead of relying on manual notes or ad-hoc shell scripts, infrastructure components are declared in code and maintained through version control. This approach is especially valuable in environments with limited IT staff, as it reduces the learning curve and allows easier recovery in case of failure.

DevSecOps and Embedded Security

Security has increasingly become a core concern in automated environments. Kommneni et al. (2024) describe the DevSecOps methodology, where security is embedded into all phases of development and infrastructure deployment [13]. They argue that relying on traditional security checks after deployment is no longer sufficient. Instead, security controls such as secrets encryption, automated testing, and auditing must be built into provisioning and orchestration pipelines.

This thesis reflects these principles through the integration of secrets management solutions, role-based access control with Keycloak, and automated TLS certificate provisioning via Traefik and Let's Encrypt. These choices align with recommended practices, as they reduce human error and help enforce consistent security policies across services.

Furthermore, the deployment of HashiCorp Vault within the infrastructure extends the security model beyond what is found in most comparable setups. Vault introduces dynamic secrets, access policies, and audit logs, features often absent in smaller environments but increasingly necessary to meet compliance and opera-

tional security expectations. Raghu (2025) demonstrates the use of Vault in financial service environments, emphasizing its ability to manage high volumes of secrets in distributed systems securely [11]. While this thesis operates on a smaller scale, the architectural principle centralizing secrets and limiting access is directly applicable.

Reverse Proxy, Monitoring, and Modular Design

Other sources stress the importance of modular architecture and secure service exposure. As discussed by Du and Nie (2011), using SSL-based reverse proxies protects public-facing services and simplifies security management by concentrating TLS termination and routing logic into a central entry point [12]. In this infrastructure, Traefik handles this responsibility, routing and securing access to internal tools such as Keycloak, Snipe-IT, and Vault. This minimizes attack surface and improves observability.

Moreover, modular design is encouraged throughout modern DevOps literature. Independent services (e.g., asset management, user directory, secrets store) are isolated in containers and orchestrated through Docker Compose. This separation of concerns allows independent testing, simplified updates, and reuse in other deployments qualities cited as essential in both academic and operational research.

3 Development Environment and System Overview

This chapter presents the technical foundation and internal environment used to design, test, and validate the infrastructure. It describes the development platform (Proxmox), provisioning techniques (Terraform, cloud-init, Debian templates), reverse proxy and DNS design (Traefik and local DNS), and supporting services for documentation. These form the building blocks upon which the infrastructure was implemented.

Rather than detailing service-specific configurations or security models, this chapter emphasizes the development and staging environment that supports modular, repeatable automation. These general-purpose setups were refined into a full production-aligned architecture, which is addressed in the next chapter.

3.1 Proxmox Virtualization Environment

Proxmox Virtual Environment serves as the foundational layer for the infrastructure developed during this thesis. It is an open-source virtualization management platform that combines Kernel-based Virtual Machine (KVM) for full virtualization and LXC containers for lightweight deployments. One of its key strengths is the web-based interface, which simplifies the creation, configuration, and monitoring of VMs and containers.

In this project, Proxmox hosts several core components of the infrastructure. These include:

- A Debian cloud-init template used to provision VMs automatically through Terraform.
- A dedicated VM running Docker and Docker Compose to manage container-based services such as Keycloak, Snipe-IT, and HashiCorp Vault.
- Supporting infrastructure such as the reverse proxy with Traefik and DNS configurations.

The choice of Proxmox was driven by its flexibility, cost-effectiveness, and ease of integration with IaC workflows. It provides an ideal environment for building a reproducible, modular infrastructure without vendor lock-in. Its support for cloud-init is particularly important, as it enables automated VM provisioning with custom configurations, making the infrastructure setup fully scriptable and repeatable.

Integration with Terraform

Terraform interacts with the Proxmox API to define VMs as code. Each machine is described with its resources, template, and network configuration. Combined with cloud-init, this allows a new VM to be instantiated and initialized in a predictable and automated manner. The full provisioning workflow is illustrated in Figure 3.1, showing how Terraform communicates with Proxmox and cloud-init to deliver fully configured instances.

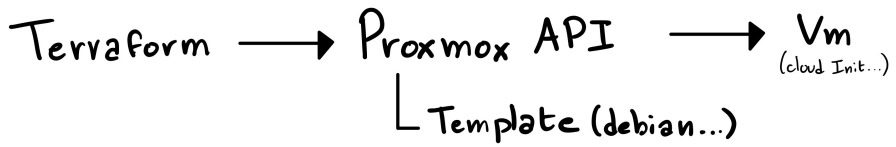

Figure 3.1: Provisioning Workflow from Terraform to Proxmox with Cloud-Init

This automation reduces human error, ensures consistent environments, and accelerates deployment. For example, setting up a new Docker host or a test environment can be done with a single command.

Performance and Reliability

According to a comparative study by Algarni et al. [15], Proxmox demonstrates efficient performance when compared to other hypervisors like Xen and standard KVM. Their benchmarks showed that Proxmox maintained low virtualization overhead, solid memory and CPU performance, and reliable I/O throughput across different operating systems. These characteristics are essential when hosting multiple services that require consistent uptime and responsiveness.

Backup and Maintenance

Proxmox also includes a built-in backup system that supports full and incremental snapshots of VMs. This feature provides a simple but effective way to secure data and recover from configuration errors or system failure. While not fully integrated into the setup yet, the Proxmox Backup Server is planned for future use to support encrypted, deduplicated backups on a scheduled basis.

Overall Role in the Infrastructure

The Proxmox host acts as the core engine of virtualization, allowing all major components of the infrastructure to be isolated into different virtual environments while maintaining centralized control and visibility. This setup provides strong security boundaries, simplifies resource management, and enables efficient experimentation and testing without risking the integrity of the rest of the system.

An overview of the full Proxmox-based infrastructure is presented in Figure 3.2, showing the separation between the template system and production containers.

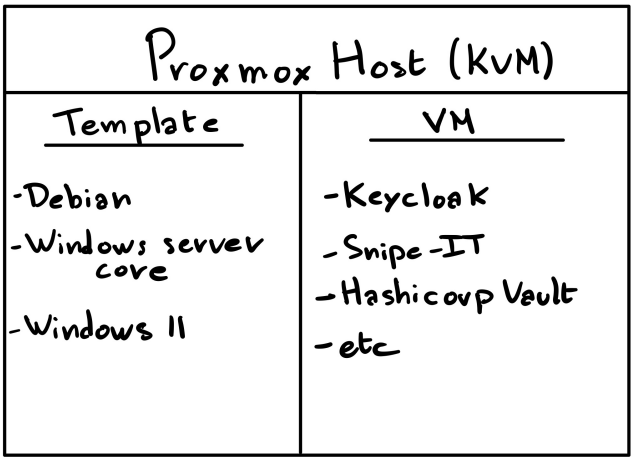

Figure 3.2: High-Level View of Infrastructure Hosted on Proxmox

By using Proxmox, the infrastructure gains a solid and scalable base that can be extended, maintained, and automated using open-source tools without compromising performance or maintainability.

3.2 DNS and Traefik-Based Reverse Proxy Architecture

A well-structured DNS and reverse proxy setup is a core requirement in any modern infrastructure. It improves user experience by providing simple, consistent URLs, enforces secure connections via TLS, and streamlines routing as services scale or change. In this infrastructure, DNS entries and a Traefik-based reverse proxy collaboratively ensure containerized services are accessible, secure, and easy to maintain.

DNS Configuration

Each deployed service such as `keycloak.company.local`, `vault.company.local`, and `snipeit.company.local` is assigned within the local DNS zone to resolve to the IP of the Debian VM hosting Traefik. This centralized DNS mapping simplifies service discovery and enables transparent migration of the proxy host without requiring changes to upstream records.

Traefik as Reverse Proxy

Traefik is a modern, cloud-native reverse proxy and load balancer designed for dynamic infrastructure. It plays a central role in this setup by securely managing incoming HTTP/HTTPS traffic and distributing it to the appropriate backend containerized services. Unlike traditional reverse proxies that rely on static configuration files, Traefik dynamically discovers services by listening to container events from Docker and adjusting its routing rules on-the-fly.

Deployed as a Docker container itself, Traefik is configured using environment variables and service labels defined within each service's `docker-compose.yml`. This label-driven mechanism eliminates the need for manual updates or reloading configuration files when services are added, removed, or modified. As a result, developers and administrators benefit from faster iteration, simplified operations, and reduced human error.

One key advantage of Traefik is its native integration with Let's Encrypt, which is used here to automatically issue and renew TLS certificates. This guarantees HTTPS support across all exposed services with no manual certificate handling. Middleware features also allow for policy enforcement such as automatic HTTP-to-HTTPS redirection, request header forwarding, and rate limiting, further strengthening both usability and security.

In this infrastructure, Traefik acts as the single entry point for external access to

services like Keycloak (identity provider), Vault (secrets manager), and Snipe-IT (asset management). Each service is assigned a subdomain (e.g., `vault.company.local`, `keycloak.company.local`), and Traefik ensures traffic is routed correctly based on hostname. Internally, containers are isolated across separate Docker networks, and only Traefik is directly exposed to clients.

This architectural design not only enhances modularity but also reinforces the security perimeter, as backend services are shielded from direct exposure. Figure 3.3 illustrates the role of Traefik in the infrastructure, sitting between DNS resolution and the containerized services it proxies.

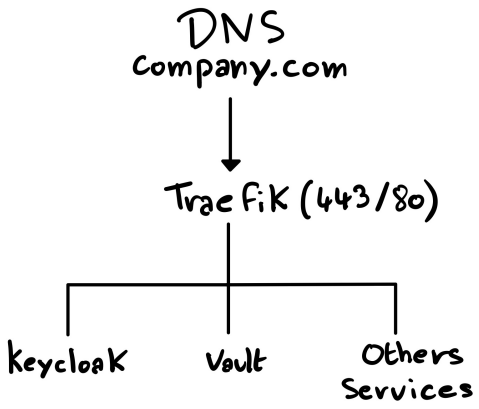

Figure 3.3: DNS and Reverse Proxy Routing Through Traefik

By consolidating access control, TLS management, and service discovery, Traefik simplifies the deployment of scalable and secure microservice environments an essential requirement for SMEs aiming to reduce overhead without compromising security or maintainability.

Performance in Container Environments

A study comparing load balancers like HAProxy, NGINX, Envoy, and Traefik found that Traefik performs competitively, offering unexpected efficiency in containerized environments thanks to its native integration with orchestration systems **diva2020load**. This justifies its choice in dynamic setups.

TLS Termination and Let's Encrypt

Automatic TLS certificate management is enabled through Let's Encrypt integration. Traefik handles certificate issuance, renewal, and deployment without admin effort. The scalability and reliability of Let's Encrypt have been validated in large deployments, highlighting the importance of automated certificate infrastructure [16].

Middleware and Security

Traefik's middleware capabilities allow for:

- Automatic HTTP → HTTPS redirection
- Enforcement of hostname-based routing
- Forwarding of headers like `X-Forwarded-For` and `Host`

These controls provide central enforcement of security and logging policies before requests reach backend services.

Scalability and Maintainability

This DNS + Traefik architecture maximizes maintainability. Adding or updating a service only requires editing its Docker Compose labels and DNS entries. There's

no need to modify Traefik's configuration, making it ideal for environments where simplicity and adaptability are priorities.

Overall, this approach delivers a clean, secure gateway for container-based services, balancing operational simplicity with strong access control.

3.3 Use of Cloud-Init and Debian Templates

To streamline the provisioning of VMs in Proxmox, this infrastructure uses a combination of cloud-init and preconfigured Debian templates. This approach significantly reduces manual setup time, ensures consistency across virtual environments, and enables automation from the very beginning of a VM's lifecycle.

What is Cloud-Init?

Cloud-init is an industry-standard tool used for initializing cloud and VM instances. It runs early in the boot process and allows dynamic injection of configurations such as hostname, SSH keys, user accounts, or even package installation. Proxmox supports cloud-init directly via a special drive in the `nocloud` format.

In this setup, each VM template is prepared with cloud-init support so that network configuration, user credentials, and SSH access are defined automatically at the time of deployment. This not only saves time but also eliminates variability between VMs that could lead to unexpected behavior or security gaps.

According to its official documentation, cloud-init is widely used across major platforms such as OpenStack, AWS, and Azure, making it a reliable and well-supported tool for infrastructure automation[1].

[1] `https://cloudinit.readthedocs.io/en/latest/`

Building Debian Templates

Debian is selected as the base operating system due to its reputation for stability, long-term support, and mature packaging system. The official Debian site explains that Debian is valued for its stable releases, predictable upgrade process, and wide architectural support [17]. This makes it a preferred choice for server environments where reliability and consistent behavior are essential. The Unix StackExchange community echoes this view, noting that Debian's stability guarantees ABI/API consistency across point releases a crucial property for systems that require long-term compatibility **debianStable**.

Templates are prepared using the following steps:

1. Install Debian on a clean VM.
2. Add the `cloud-init` package and enable corresponding systemd services.
3. Apply basic configurations (e.g., disable root login, update system packages).
4. Shut down the VM and convert its disk to a Proxmox template.

Once the template is available, Terraform references it during VM creation. Combined with cloud-init, this produces near-instant, reproducible VM provisioning with minimal manual intervention.

Why Debian Matters

Templates guarantee a consistent baseline for all infrastructure nodes. Cloud-init brings in dynamic configuration, enabling customization at runtime. This separation of base image and runtime customization supports a clean, modular architecture that aligns well with DevOps and Iac principles.

Using Debian as the base OS and integrating cloud-init improves maintainability, reduces variability across environments, and supports long-term system stability all vital for SME infrastructures.

Architecture Illustration

The interaction between Terraform, Proxmox, and Cloud-Init is shown in Figure 3.4. This visual highlights the sequential logic involved in provisioning a new VM, from template to fully configured instance.

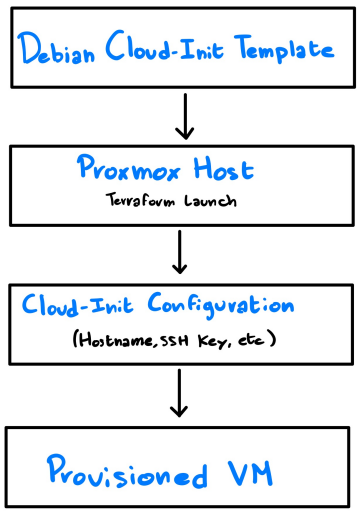

Figure 3.4: Automated provisioning pipeline using Terraform, Proxmox, and Cloud-Init

3.4 Internal Documentation with Outline

Internal documentation is a cornerstone of reliable, maintainable, and scalable infrastructure. In DevOps-oriented environments, documentation not only records what a system does it also supports collaboration, knowledge sharing, onboarding, and

continuous improvement. For this infrastructure, a documentation-first approach was adopted using Outline, a modern team knowledge base that helps ensure all decisions, implementations, and processes remain transparent and accessible.

Why Document Internally?

Research shows that organizations with strong internal documentation practices are significantly more effective in software delivery and team coordination. According to OpsLevel, well-maintained documentation improves developer experience, facilitates onboarding, and reduces the risk of configuration drift or undocumented shortcuts [18]. Similarly, the DevOps community consistently highlights documentation as a pillar of high-performance teams, tying it to metrics like deployment frequency, change failure rate, and recovery time [19].

In the context of this infrastructure, documentation plays a practical role in ensuring that Iac remains understandable and reusable over time. From the way services are exposed via Traefik labels, to how Ansible playbooks are encrypted and structured, every component has a written explanation stored in the Outline platform.

Using Outline for Team Collaboration

Outline was selected for its simplicity, support for markdown syntax, real-time editing, and intuitive search interface. Documentation is organized by component: for instance, Terraform provisioning has its own dedicated page, while services like Keycloak and Vault are documented with setup instructions, usage notes, and update history. Rather than relying on ad-hoc documents or file-based wikis, Outline provides a consistent and searchable platform that is easy to navigate and maintain.

Each documentation entry includes a brief introduction, architectural explanation, and description of relevant steps such as deployment instructions or trou-

bleshooting procedures. This standardized format ensures that new team members can quickly understand the infrastructure without requiring one-on-one knowledge transfers or guesswork.

Embedding into the Workflow

To avoid outdated documentation, updating Outline is integrated into the regular deployment process. Whenever a new service is added, modified, or redeployed, the corresponding documentation page is reviewed and updated. This habit is reinforced by including documentation reminders in pull request templates, ensuring that changes to infrastructure code are always accompanied by revised instructions or diagrams.

Outline's version control and change history allow tracking of who made what changes and when useful during debugging or audits. Over time, this approach transforms documentation into a live, evolving part of the system instead of a static afterthought.

Documentation as Part of DevOps Culture

Documentation in this infrastructure is more than just a support tool, it reflects the values of the DevOps methodology. DevOps emphasizes feedback, collaboration, and continuous learning, and documentation is one of the key interfaces through which these values are realized.

By maintaining accurate, modular, and living documentation, teams can foster a shared understanding of their systems. This supports onboarding, enables quicker incident response, and lowers the risk of knowledge silos. Outline becomes a single source of truth not just for technical processes, but also for shared norms around how infrastructure is developed and maintained.

The practice of documenting alongside development reinforces the principle that

good documentation is not a separate task it is part of building the infrastructure itself. This alignment between tooling, team practices, and infrastructure design contributes to long-term system reliability and team efficiency [19].

4 Design and Architecture of the Automated Setup

Building on the environment described in Chapter 3, this chapter shifts focus to the architectural model used to organize, deploy, and secure the automated infrastructure. While Chapter 3 discussed the foundational elements like virtualization and provisioning tools, this chapter defines how those components were assembled into a functional system.

The infrastructure described here follows a modular, layered structure using open-source technologies to orchestrate virtualization, containerization, access control, and secrets management. Each design decision emphasizes automation, reproducibility, and operational security to simulate a real-world deployment pipeline suitable for SMEs.

The following sections outline how key services like Keycloak, Snipe-IT, and Vault were integrated into the infrastructure, how AD was deployed for product testing, and how network and security boundaries were defined using IaC principles.

4.1 Infrastructure Topology and Service Map

The infrastructure designed in this work is tailored for SMEs aiming to build reproducible, secure, and maintainable systems using open-source tools and automation. The overall design follows a modular and layered approach, separating concerns

between virtualization, service provisioning, and application management.

At the foundation is a Proxmox VE instance, serving as the hypervisor for hosting VMs. Proxmox was selected for its open-source model, mature web-based interface, and robust support for KVM-based virtualization. On top of this layer, the infrastructure provisions Debian-based VMs using Terraform, taking advantage of preconfigured cloud-init templates to automate initial configuration and networking.

One of these VMs serves as a central host for containerized services. Within this Docker-enabled environment, essential services are deployed via Docker Compose, including:

- **Keycloak** – Identity and access management.
- **Snipe-IT** – Asset tracking and lifecycle management.
- **HashiCorp Vault** – Secure secrets management.
- **Traefik** – Reverse proxy and TLS termination.

This approach centralizes orchestration and resource monitoring while maintaining security and logical separation at the container level. Each service is made accessible through domain-based routing managed by Traefik, with DNS entries pointing to the proxy host. Let's Encrypt handles TLS certificate provisioning and renewal, enabling encrypted communication with minimal overhead.

Additionally, an AD test environment was provisioned in a separate Windows Server VM. This internal directory service is isolated from production traffic and is used to validate integration of the organization's product with enterprise IAM systems. The AD VM shares the same virtual network, allowing secure, real-world-like testing scenarios.

The full architecture is shown in Figure 4.1 for a visual representation of the service map.

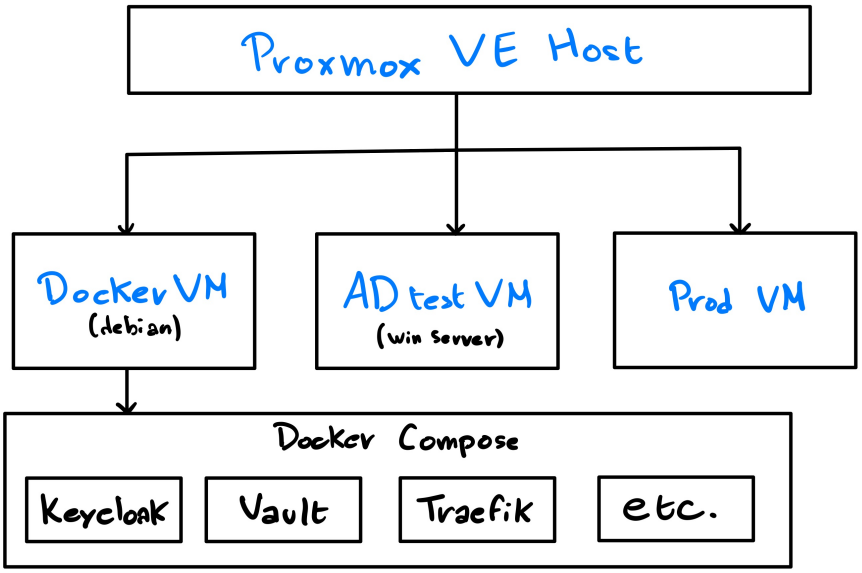

Figure 4.1: Infrastructure topology showing VMs and containerized services

This layered and modular design aligns with best practices described in the literature. Koneru [8] emphasizes reproducibility and automation using tools like Terraform, advocating IaC approaches over manual configuration or commercial alternatives such as AWS CloudFormation. Özdoğan et al. [20] recommend separating infrastructure provisioning from application state through layered abstractions. Dasari [21] supports modular, extendable designs that facilitate ongoing evolution and integration with CI/CD or observability tools.

Rather than prescribing a one-size-fits-all architecture, this thesis presents a practical blueprint rooted in flexibility and simplicity. The implementation serves as a proof of concept for reproducible infrastructure that can evolve with organizational needs.

4.2 Active Directory for Internal Product Testing

A core component of the infrastructure described in this thesis is the deployment of a dedicated, isolated AD environment. This environment was created with the purpose of testing how the company's product interacts with AD-based systems, mimicking the kinds of environments typically found in client infrastructures. Such testing is essential for understanding authentication workflows, directory integration, permission handling, and group-based policy enforcement.

The AD environment was deployed on a Windows Server 2022 VM hosted in Proxmox. The provisioning and configuration were entirely automated using Terraform and Ansible. Terraform was responsible for defining the virtual resources such as CPU, memory, networking, and disk allocation, while Ansible handled the installation and configuration of the AD domain services. This setup allowed for repeatable and consistent deployment, reducing manual setup time and the risk of misconfiguration.

To ensure sensitive information such as administrator passwords remained secure throughout the automation process, Ansible Vault was used. This tool encrypts variables and secrets, preventing them from being exposed in plain text within the repository or playbooks. By applying secure credential management from the beginning of the provisioning process, the infrastructure aligns with modern best practices for DevOps security.

Isolated AD environments like this are widely used in development and operations teams to test integration features without affecting production systems. As highlighted by Adam Bertram (2018), testbed AD setups help ensure test-production parity, prevent unintended changes to live systems, and allow automated configuration scripts to be validated in a controlled setting [22]. This mirrors the strategy used in the company experience, where the goal was to build a safe, realistic, and resettable environment for product validation.

The approach also follows guidance from Madlabber (2019), who demonstrated how to build AD with automation tools and scripting logic [23]. This aligns well with the use of Ansible in this project.

Finally, the security implications of testing AD environments should not be overlooked. As Mokhtar et al. (2022) discuss, AD remains a primary target for attackers due to its centralized role in identity and access control [24]. The isolated test setup used in this thesis allowed the team to evaluate the product's behavior in various AD scenarios without exposing real credentials or production systems. This also enabled controlled simulations of common attacks and misconfigurations, reinforcing both product security and internal readiness.

Overall, the deployment of the AD environment represents a practical example of how IaC, automation, and security practices converge to support development needs. While the specific goal was internal product testing, the architectural approach, tooling, and security posture offer a transferable blueprint for similar use cases in other organizations.

4.3 Snipe-IT and Keycloak Deployment Model

Two key services in the infrastructure Snipe-IT and Keycloak were deployed as Docker containers managed via Docker Compose. These components address two essential needs of modern IT environments: internal asset lifecycle management and centralized identity and access control. They were integrated behind a Traefik reverse proxy with TLS via Let's Encrypt and mapped under custom DNS names, allowing secure, maintainable, and accessible deployments.

Snipe-IT Deployment and Asset Traceability

Snipe-IT is an open-source web application designed for comprehensive IT asset management. In this infrastructure, it was deployed via Docker Compose, with its configuration handled through Ansible. The application runs on a Debian-based VM provisioned using Terraform and hosted on the Proxmox virtualization platform. It interfaces with a MariaDB backend, and secrets such as database credentials and SMTP tokens are stored securely using Ansible Vault to prevent leakage of sensitive information.

From a management standpoint, Snipe-IT provides a centralized system to catalog all physical and digital assets within the company. It enables tracking of inventory like laptops, phones, licenses, accessories, and user assignments, which is essential for operational visibility. Each item in the database includes ownership information, location, warranty status, and lifecycle stage making inventory audits faster, more accurate, and far less reliant on manual tracking methods. This is particularly valuable in environments where remote work or hardware rotation is common.

Beyond inventory management, Snipe-IT plays a strategic role in the cybersecurity framework of the infrastructure. By maintaining a real-time map of which user holds which asset, the system reinforces the principle of accountability. When a security incident arises such as a data breach or compromised device the database can be quickly queried to identify which endpoint may have been involved and who had custody at the time.

To further strengthen its security utility, a custom field was introduced to track the encryption status of each workstation. This enhancement supports compliance with data protection policies such as GDPR or ISO/IEC 27001, where encryption of endpoints containing sensitive or personal data is often a requirement. It also allows the IT team to spot gaps in data-at-rest protection, helping them prioritize

remediation actions and reduce the potential impact of device loss or theft.

Additionally, asset data from Snipe-IT can be leveraged during risk assessments and patch management. When new vulnerabilities are published (e.g., via CVEs), the asset database can be queried to determine which machines or systems might be affected based on their hardware model, operating system, or installed software. This enables more targeted, efficient response during security patch cycles.

In summary, the inclusion of Snipe-IT transforms asset tracking from a logistical necessity into a proactive security control. By combining lifecycle management with visibility into physical device security states, it creates a reliable foundation for both operational management and cybersecurity resilience.

Keycloak Deployment and Centralized Identity Management

Keycloak was deployed as a containerized service on the same Proxmox host and integrated into the infrastructure using Docker Compose. It acts as the central Identity Provider (IdP) across all internal services, replacing fragmented login systems with a unified Single Sign-On (SSO) approach. Keycloak was configured with custom realms and clients for services like Snipe-IT, Outline, and other internal tools.

By using protocols such as OpenID Connect (OIDC) and SAML 2.0, Keycloak enables secure authentication and delegation across web-based services. The SSO model reduces password fatigue and simplifies access management: users authenticate once via Keycloak and are granted access to all authorized services without needing to re-enter credentials. This not only improves user experience but also decreases the attack surface exposed by weak or reused passwords.

Keycloak was selected not only for its extensibility and modern interface, but also for its alignment with recognized security best practices. OWASP's AppSensor project and its Application Security Verification Standard (ASVS) emphasize

centralized authentication as a method to reduce implementation errors across applications and increase visibility into access control events. Keycloak supports fine-grained access control via RBAC, policy-driven authentication, and even step-up authentication for sensitive workflows.

Security protections in Keycloak include:

- Support for multi-factor authentication (MFA) using TOTP or external identity providers.
- Configurable password policies and brute-force detection.
- Integration with LDAP or AD for federated identity.
- Administrative logging and session management for compliance.

Additionally, Fett et al. [25] performed an in-depth formal security analysis of OpenID Connect the protocol underlying most Keycloak interactions and demonstrated its strong security guarantees when properly implemented. Li et al. [26] further discuss potential vulnerabilities in OAuth and OIDC flows, such as CSRF attacks on login endpoints, and outline mitigations that are considered in Traefik's header-based routing setup. Meanwhile, the capabilities and practical deployment models of Keycloak were positively evaluated in a review by Divyabharathi and Cholli [27], highlighting its value for secure IAM in microservice environments.

In this infrastructure, Keycloak not only secures access to critical tools, but also forms the backbone of a scalable, policy-compliant authentication architecture that can evolve with organizational growth and security needs.

4.4 Secrets Management with HashiCorp Vault

Managing sensitive data is a fundamental requirement for secure infrastructure design. In modern automated environments, exposing secrets such as database cre-

dentials, API tokens, or private keys can lead to serious security incidents. To avoid these risks, the infrastructure setup integrates HashiCorp Vault a robust, open-source tool designed for centralized secret storage, encryption, and access control.

Vault Deployment Setup

In this project, Vault was deployed as a containerized service using Docker Compose, hosted on a Debian VM provisioned with Terraform within the Proxmox hypervisor. This allowed for tight control over the environment while maintaining flexibility for future changes. Vault was initialized using Shamir's Secret Sharing mechanism, distributing the unseal keys among trusted holders to prevent any single point of failure or unauthorized access. Access to Vault is restricted to internal networks, routed securely through the Traefik reverse proxy, and firewalled from external systems.

Secrets were organized within Vault's key-value engine, where credentials for services such as Snipe-IT, Keycloak, and other internal applications were securely stored. Rather than embedding these credentials into playbooks or scripts, the infrastructure used runtime retrieval, fetching secrets directly from Vault during provisioning. This approach aligns with the practices described by Basak et al. (2022), who emphasized the benefits of centralizing secrets to improve maintainability, access control, and compliance with secure development standards [28].

GitLab Integration for Deployment Approval

One of the most advanced integrations of Vault within this setup is its connection to the deployment pipeline managed by GitLab. In practice, this allows the Chief Technology Officer (CTO) to cryptographically authorize the release of production workloads. The Vault instance controls the signing process required to unlock the GitLab pipeline for production deployment. This ensures that even if an unauthorized push occurs within GitLab, no deployment can be triggered without explicit approval.

Such an approach embeds human control into an automated DevOps environment, significantly increasing the integrity of the CI/CD pipeline. This methodology is consistent with the work published in the International Journal of Science and Research (2021), which advocates for integrating secret management tools directly into pipeline workflows to improve automation without sacrificing control [29].

Audit Logging and Security Guarantees

Vault's auditing and logging capabilities also contribute to the security posture of the infrastructure. Every interaction with the Vault server is logged and timestamped, allowing for traceability and historical analysis. These logs are especially useful for tracking changes in access policies, credential use, and unexpected or unauthorized access attempts. Raghu (2025) emphasized Vault's utility in environments that demand both cryptographic enforcement and compliance monitoring, such as financial services and enterprise deployments [11].

By integrating HashiCorp Vault into the broader automation and provisioning stack, this infrastructure benefits from a security-first design where secrets are neither exposed in plaintext nor stored statically. Instead, they are securely distributed on demand, tightly scoped through ACL policies, and monitored continuously. This provides a significant step forward in creating an auditable, compliant, and resilient infrastructure capable of supporting secure operations at scale.

4.5 Security Design Principles (Least Privilege, Encrypted Backups, Isolated Access)

Modern infrastructure security demands an approach that integrates protection mechanisms into the foundation of the system, rather than treating them as afterthoughts. This section outlines the security principles applied during the infras-

tructure setup, focusing on minimizing attack surfaces, ensuring data availability, and maintaining controlled access throughout.

Least Privilege and Identity Scoping

A central concept implemented is the principle of least privilege. All services and users are configured to operate with only the permissions necessary to perform their intended function. This design choice significantly reduces the blast radius of potential breaches and misconfigurations.

In practice, service accounts were scoped with restrictive permissions. For instance, the HashiCorp Vault deployment grants read/write access only to secrets relevant to each service, and provisioning scripts used during deployment are executed with temporary credentials. Network-level controls also reinforce this principle, ensuring no unnecessary ports or interfaces are exposed.

The infrastructure leverages centralized identity management through Keycloak, enabling unified user authentication and granular role-based access controls (RBAC). These access controls are crucial to prevent privilege escalation and to isolate administrative access from everyday operations.

Encrypted Backups and Data Resilience

Protecting data at rest is vital, especially in environments where infrastructure is heavily automated. To safeguard the VMs and service states hosted on Proxmox, encrypted backup routines are scheduled using the Proxmox Backup Server. These backups are not only encrypted but also incremental, reducing storage usage while preserving historical data points for recovery.

This design aligns with the recommendations outlined by the National Institute of Standards and Technology (NIST) in their guidelines for secure storage infrastructures [30]. According to their framework, ensuring data confidentiality, integrity, and

availability requires multi-layered security approaches, including encryption, access controls, and resilient recovery procedures all of which were applied in this deployment.

Furthermore, the integration with GitLab CI/CD enables a secure release process where the CTO signs each production deployment pipeline using Vault-provided credentials. This digital signature enforces accountability while preventing unauthorized modifications to the deployment pipeline.

Isolated Access Through Container Segmentation

Another key architectural principle is the isolation of services using container-level segmentation. Docker networks are configured to restrict inter-container communication to only what is explicitly required. For example, Vault and Keycloak are placed in separate internal-only networks, inaccessible from the internet, and not reachable by unrelated services. This containment drastically reduces the potential for lateral movement during an attack.

The importance of container segmentation is emphasized by Thiyagarajan and Nayak (2025), who outline best practices for runtime container hardening and Docker security. Their study shows that network-level boundaries, combined with container runtime restrictions, offer a robust defense against common attack vectors in cloud-native environments [31].

Further reinforcement of this approach comes from Al-Ofeishat and Al-Shorman (2024), who advocate for segmentation and micro-segmentation as key strategies to enhance control over east-west traffic in virtualized infrastructures [32]. In the implementation described in this thesis, containers are grouped into logically separated networks based on functionality, with strict ingress and egress rules. This prevents unauthorized services from reaching sensitive components and allows for fine-grained security policies to be enforced at the network level.

Embedded Secure Design Principles

The infrastructure was designed with adherence to OWASP security principles, particularly those outlined in the OWASP Secure Coding Practices and OWASP Security by Design guidelines. Specifically, secure defaults, defined trust boundaries, audit logging, and layered defenses were embedded throughout. For example, the reverse proxy enforces HTTPS by default, all configuration files are version-controlled, and secrets are rotated regularly.

These principles not only reduce risks but also foster a proactive security culture, wherein infrastructure evolves with the assumption that breaches can occur and therefore must be anticipated, detected, and mitigated swiftly.

5 Implementation and Validation

This chapter presents how the designed infrastructure was translated into a working environment and how its reliability and security were evaluated. The focus is on the practical steps taken to implement the system using Iac tools, containerization, and automation workflows. It also describes the validation procedures applied to confirm functionality, security properties, and maintainability. Together, these sections provide a comprehensive view of how the conceptual architecture was realized in practice and how its effectiveness was assessed.

5.1 Provisioning with Terraform and Ansible

A two–layer Infrastructure-as-Code workflow was adopted to move from design to an operational environment. Terraform provides the declarative "blueprint" of resources VMs, storage volumes, VLANs while Ansible performs the procedural, idempotent configuration of each instance after creation. Using these tools in tandem satisfies three core goals: reproducibility, modularity, and auditable change control.

Terraform-Driven Infrastructure Definition

Terraform modules define every Proxmox resource, from cloud-init Debian templates to network bridges and firewall rules. Each VM is parameterised through variables so that staging and production environments differ only by a handful of values in a `tfvars` file. Remote state is stored in a version controlled backend, ensuring the

provisioning team has a single source of truth. Outputs primarily VM IP addresses, hostnames, and SSH fingerprints are exported automatically in JSON format to feed downstream automation.

Dynamic Inventory and Ansible Bootstrapping

Ansible consumes Terraform's JSON outputs as a dynamic inventory, guaranteeing that only what Terraform created is ever targeted for configuration. The initial bootstrap playbook installs base security controls (unprivileged users, SSH hardening, and package signing keys) and the Docker Engine. Subsequent plays deploy Snipe-IT, Keycloak, and Vault via Docker Compose, set up Traefik routing, and register DNS records. Because each play is idempotent, repeated runs simply enforce drift correction an essential property for CD pipelines.

Security and Compliance Integration

Secrets required during provisioning database passwords, TLS keys, GitLab runner tokens are fetched at runtime from HashiCorp Vault using Ansible lookups, never appearing in plaintext or in Terraform state. This design meets OWASP guidance on secure secret handling and aligns with zero trust assumptions in DevSecOps pipelines. Least-privilege service accounts restrict both Terraform and Ansible to the minimal scope needed, any attempt to exceed these roles is denied and logged.

Validation and Continuous Improvement

Continuous Integration (CI) in GitLab runs `terraform validate`, `terraform plan`, and Ansible's `ansible-lint` on every merge request. These steps ensure that code changes conform to syntax and security standards before being merged into the main branch. Plans are reviewed by peers and validated against predefined infrastructure policies to ensure correctness and maintain traceability of infrastructure

modifications.

Post-deployment acceptance tests are automated using `molecule` and `testinfra`. These tools verify key service properties such as health checks, network port exposure, and compliance with security controls, helping detect misconfigurations early in the lifecycle.

This layered approach follows the reference architectures documented by Thota et al. (2020), who emphasize the use of Terraform for baseline provisioning and Ansible for higher-order configuration [33]. Pathak (2024) similarly demonstrates that separating structural and procedural logic reduces cognitive load, simplifies code review, and accelerates recovery from failure [34].

5.2 Dockerized Service Deployments

The infrastructure makes extensive use of Docker to deploy key internal services such as Snipe-IT, Keycloak, and HashiCorp Vault. Each service is encapsulated within a container, providing portability, modularity, and reproducibility across environments. These containers are orchestrated using Docker Compose on a Debian VM provisioned through Proxmox and automated with Terraform and Ansible.

Service Isolation and Multi-Container Setup

Each container is defined using ‘docker-compose.yml‘ files that specify services, environment variables, networks, and volumes. This enables logical separation between services while maintaining shared network channels only where necessary. The setup also uses Compose profiles to handle environment-specific deployments for staging and production. This structure improves management and reduces deployment complexity.

As Diekmann et al. (2019) emphasize, effective access control between container-

ized services is crucial. The use of Docker-defined bridges and segmentation policies ensures that only authorized services communicate with each other [35].

Security Hardening in Container Environments

Security measures include the use of non-root container users, restricted filesystem access, and TLS-encrypted communication between services. HashiCorp Vault is used to inject secrets securely at runtime, while Traefik handles SSL termination and traffic routing through secure reverse proxies. Secrets are never stored in plaintext on disk or environment variables.

Following the recommendations by Thiyagarajan and Nayak (2025), container hardening includes disabling unused capabilities, minimizing base images, and using automated vulnerability scans on Dockerfiles and images before deployment [31].

Containers on VMs, Not LXC: Production Considerations

Although Proxmox supports LXC containers, Dockerized services are deployed inside full KVM-based Debian VMs. While LXC containers offer better performance and resource usage, they have limitations in terms of security isolation and kernel compatibility. For production systems, prioritizing stability and isolation is more important than minor performance gains.

Using full VMs allows stricter access control, better resource isolation, and independent OS-level management. If one container or service stack fails, it does not jeopardize others. In critical systems, this trade-off ensures recoverability and mitigates attack surfaces a particularly important factor in infrastructures handling sensitive data or external access.

Proxmox-Driven Service Separation

Proxmox enables distribution of services across multiple VMs when scaling up. For instance, Snipe-IT and Vault could be deployed on separate VMs for increased isolation and performance monitoring. This segmentation enhances disaster recovery, simplifies troubleshooting, and allows tailored backup policies for each service group.

By combining Proxmox's virtualization control with Docker's lightweight service design, the infrastructure gains flexibility, security, and scalability. This layered approach ensures that services are isolated both at the container and system levels, while still being easy to redeploy or replicate through Iac techniques.

5.3 DNS and Reverse Proxy Configuration

In a microservices-oriented infrastructure, effective traffic management and secure routing are critical. This project employed a DNS-driven design combined with the Traefik reverse proxy to manage service exposure, enforce encryption, and streamline traffic control. The architecture leverages both static and dynamic routing to accommodate containerized services within a Docker host as well as services running across VMs.

Traefik as a Reverse Proxy

Traefik serves as a Layer 7 (application layer) reverse proxy and load balancer. It was chosen for its native support of dynamic service discovery through Docker integration, as well as its static configuration capabilities for external services. In this setup, containerized applications such as Keycloak, Vault, and Snipe-IT are dynamically routed based on metadata defined in their `docker-compose.yml` files. For services running outside the Docker host such as additional VMs or external tools Traefik uses a static configuration file to define routing rules and TLS settings.

This hybrid configuration model supports flexible service deployment while preserving strict access control and consistency across the network layer.

DNS Infrastructure Using NixOS

Rather than relying on a cloud-based DNS provider such as Cloudflare, the infrastructure utilizes NixOS's built-in DNS services. A local authoritative DNS server is configured to manage internal domain resolution for subdomains such as `keycloak.company.local`, `vault.company.local`, and `snipeit.company.local`. These records map to the IP address of the VM running Traefik.

By hosting the DNS service in-house, the system avoids reliance on external providers for domain resolution, enhancing both control and privacy. DNS configuration is declarative and reproducible via Nix expressions, ensuring versioned tracking of changes and easy rollback in case of misconfiguration.

TLS Management and Security Hardening

TLS certificates are handled by Traefik using its built-in support for Let's Encrypt. Since DNS-01 challenges were not feasible with the internal DNS setup, HTTP-01 challenge was used instead. Traefik automatically issues and renews certificates for all exposed services, ensuring that HTTPS is enforced uniformly across the system.

Traefik's middleware features further strengthen the security posture. Policies such as automatic HTTP-to-HTTPS redirection, secure header injection, and request filtering are centrally managed. Internal-only APIs, such as Vault's administrative interface, are isolated on private Docker networks and are never exposed externally. This segmentation helps enforce the principle of least privilege and reduces the surface area for attacks.

Network Boundaries and Zero Trust Principles

The routing configuration was designed with Zero Trust principles in mind. Traefik is configured to explicitly route only to known, predefined services. Any request targeting an undefined route is rejected by default. Combined with the use of Docker network isolation, this ensures that services are only accessible in the context in which they are intended to be used.

Following the model proposed by Nuryasa and Suharjo (2024), SMEs infrastructures can implement robust access control without the complexity of enterprise firewall appliances by carefully structuring routing layers [36]. This approach is particularly effective in virtualized environments like this one, where isolation can be enforced via containers and hypervisors.

Benefits and Challenges of Hybrid Routing

The infrastructure benefits from a mixed routing model. Dynamic routing with Docker labels simplifies container orchestration on the main service host, while static routing ensures that legacy systems, monitoring tools, or services deployed on separate machines can still be integrated into the same namespace and security framework.

This dual-mode operation supports consistent enforcement of TLS, access logging, and authentication middleware regardless of whether the service is containerized or hosted independently. However, managing both static and dynamic sources introduces complexity, particularly in configuration synchronization and certificate handling, which must be carefully documented and automated to avoid errors.

Overall, this architecture centralizes control of service exposure, improves maintainability, and strengthens the security of the network boundary making it a strong fit for SMEs investing in infrastructure automation.

5.4 Security Measures and Hardening Steps

Securing infrastructure is an essential task, especially when automating deployments and managing multiple services across VMs and containers. In this setup, several security-oriented configurations were applied at different levels, from VM provisioning to container deployment and network exposure. This section presents the security measures that were implemented, and highlights additional best practices observed in the industry for further context.

Container Security and Deployment Isolation

To minimize the risks associated with containerized services, images were selected carefully and only trusted official or verified sources were used. Services such as Keycloak, and Vault were deployed in isolated containers using Docker Compose on a dedicated VM. Container configuration files were managed under version control, and environment variables containing sensitive information were secured using Vault.

For better separation of concerns, container services were grouped logically and hosted on different VMs when necessary. While containers can also run directly as LXC in Proxmox, Docker was run inside full VMs instead, due to the increased stability and isolation they offer in a production context. Although LXC offers better performance, full VMs are preferred in professional settings when security takes precedence over raw performance.

Thiyagarajan and Nayak (2025) emphasize that strong container security requires proper isolation, minimal images, and avoidance of privilege escalation paths, especially when used in enterprise environments [31].

Reverse Proxy, TLS, and Network Restrictions

Traefik served as the reverse proxy in the infrastructure, exposing public services (e.g., Snipe-IT, Keycloak, Vault) through HTTPS. It was configured to obtain and renew TLS certificates automatically via Let's Encrypt. This eliminates the need for manual certificate handling and ensures that all web traffic to exposed services remains encrypted.

DNS entries were configured for each service to simplify access and improve security by avoiding raw IP usage. The reverse proxy configuration also included static routing for locally grouped containers and dynamic routing for services deployed on other machines. This separation provided additional flexibility and control over service access and exposure.

Network communication between services was restricted using Docker networks and Traefik routing rules. Only required ports were exposed, and container interaction was limited to what each service strictly needed.

Secrets Management and Access Control

Sensitive secrets such as admin passwords, API keys, and database credentials were never stored in plaintext. HashiCorp Vault VaultWarden was used to manage these secrets, and access to them was limited to services and users with appropriate roles. This centralized approach to secrets management ensures better control and auditability, aligning with industry standards.

Additionally, Vault was integrated into automation workflows to make secrets retrieval dynamic and safer during deployment. As highlighted by Raghu (2025), integrating Vault into a company's internal tools enables more secure automation without exposing secrets to configuration files or repositories [11].

Security-Aware Infrastructure Choices

Beyond technical configurations, security considerations influenced key architectural decisions. Using VMs for critical services rather than running them all in containers or on a single host reduces the blast radius in case of compromise. Services were grouped logically, and provisioning with tools like Terraform and Ansible helped ensure consistent, repeatable setups with no manual misconfigurations.

Wilkens et al. (2021) recommend combining these practices with TLS decryption strategies where advanced monitoring is needed [37]. Although such monitoring was not implemented in this setup, it reflects common hardening practices for high-security environments.

Overall, these security measures contribute to a stable, maintainable, and secure infrastructure that balances automation with protection.

5.5 Evaluation of the Setup: Functionality, Security, and Reusability

To assess the effectiveness of the implemented infrastructure, three major aspects were evaluated: core functionality, security properties, and the reusability and maintainability of the configuration code.

Functionality

The infrastructure was designed to automatically deploy a working environment comprising Proxmox-hosted Debian VMs, Traefik reverse proxy, Docker-based services (Snipe-IT, Keycloak, Vault), and integrated DNS resolution. Validation was performed using acceptance tests implemented via `testinfra` and simple HTTP health checks, ensuring each service responded correctly on its expected endpoint.

Functional evaluation confirmed that service orchestration works reproducibly across deployments, and that services communicate successfully through the Traefik proxy under defined access rules.

Security Properties

Security was assessed based on observed behavior and alignment with industry best practices. Secrets were handled strictly through HashiCorp Vault, with access limited to authorized components. All external access uses TLS through Traefik, with automatic certificate issuance and renewal. Access logs and Ansible IaC code review confirmed that no sensitive data such as credentials or keys were ever stored in plaintext. Application of least privilege was verified by checking that service users and automation accounts have only the required access levels.

These practices reflect the findings of Verdet et al. (2023), who observed that many open-source Terraform deployments fail to implement security policies particularly around encryption and access controls even when they use modern IaC tools [38]. By contrast, the infrastructure described here avoids those pitfalls.

Reusability and Maintainability

Reusability and maintainability were evaluated by how easily the infrastructure code can be adapted, reviewed, and redeployed. Terraform and Ansible scripts were modularized into logical units machine templates, network configuration, service deployment facilitating environment-specific customization (development, staging, production) through simple variable substitution.

This structure aligns with Koneru (2025), who notes that high modularity, state stability, and clear CI/CD integration are key factors for enterprise IaC adoption [39]. The template-based approach ensures that new environments can be spun up with minimal adjustments, peer review of infrastructure changes becomes feasible,

and configuration drift is prevented through idempotent tooling.

Limitations and Considerations

Though the evaluation shows a functional, secure, and reusable setup, there are areas where improvements could be considered. More advanced testing of Terraform and Ansible scripts for example using automated infrastructure testing frameworks like the Automated Configuration Testing (ACT) methodology proposed by recent research was not applied in this deployment [40]. Similarly, while security smells were avoided in code, a more exhaustive static analysis (e.g. using frameworks like GLITCH) would further improve confidence in IaC hygiene.

6 Discussion

This chapter critically reflects on the infrastructure design and implementation presented in the thesis. The aim is not only to highlight achievements but also to analyze the trade-offs, challenges, and limitations encountered during the process. Particular attention is given to how the choices made affect security, usability, and scalability in practice, as well as what lessons can be learned for future improvements. The discussion also outlines potential enhancements that could make the system more resilient and better suited for production environments.

6.1 Reflections on Security, Usability, and Scalability

The development and deployment of this infrastructure provided valuable insight into the balance required between security, usability, and scalability. These three factors, while often aligned in intention, can introduce trade-offs that must be evaluated based on organizational context and available resources.

From a security standpoint, integrating tools like HashiCorp Vault and Keycloak significantly strengthened access control and secret management. The adoption of Iac reduced the risk of manual configuration errors, and the use of a reverse proxy with TLS improved communication security. Container segmentation and service isolation on Docker networks offered additional defense-in-depth measures.

However, it also became clear that security adds complexity particularly in the setup and maintenance of tools like Vault and Keycloak, which require careful planning of roles, policies, and secure storage mechanisms.

In terms of usability, tools such as Ansible and Docker Compose provided intuitive mechanisms for managing services. Once initial playbooks and compose files were written and tested, updates and redeployments became straightforward. Yet, this ease of use depends on prior investment in learning and documentation. Complex tools like Terraform or Vault present a steeper learning curve, which can affect onboarding speed for new team members or scalability in teams with limited technical expertise.

Scalability was considered from the early stages of design. The modular structure of the infrastructure deploying services as separate containers or VMs makes it adaptable to larger environments or additional features. The use of Proxmox allows for scaling horizontally across multiple physical nodes, and separating services into dedicated VMs improves fault tolerance and security. However, scalability introduces its own challenges, such as performance monitoring, log aggregation, and secrets synchronization across nodes, which would need to be addressed in a production-grade deployment.

Overall, this experience demonstrated that it is possible to build a secure and usable infrastructure that remains flexible enough to evolve. Each decision whether to isolate a service, automate a deployment, or enforce a security policy requires weighing operational simplicity against long-term maintainability and risk exposure.

6.2 Limitations and Lessons Learned

While the infrastructure developed in this thesis fulfills its objectives in terms of automation, security, and modularity, several limitations were identified throughout the implementation process.

A primary constraint was the absence of real production workloads. Although the system was tested with simulated services and identity integration, its behavior under high load or real user traffic remains unassessed. Features like centralized monitoring, logging, and alerting were not fully implemented, which limits the ability to detect performance degradation or security incidents in real time.

Another challenge was the steep learning curve of some technologies. Tools like Terraform, HashiCorp Vault, and Keycloak offer powerful capabilities, but they require substantial configuration, especially when combined. Debugging access issues in Vault policies or container network misconfigurations proved time-consuming. Additionally, certain configuration elements lacked documentation in official sources, which required manual experimentation or community forum guidance.

From a maintainability perspective, scaling the infrastructure to multiple nodes or preparing it for production migration would require added components such as configuration state management, load balancing, and high-availability setups.

Despite these limitations, this experience provided deep insight into how infrastructure decisions impact long-term stability and security. It highlighted the importance of documentation, reproducibility, and the benefit of investing time in automation to reduce manual tasks. This work also demonstrated the value of combining existing tools effectively, rather than building complex systems from scratch.

7 Conclusion and Future Work

This final chapter summarizes the main achievements of the thesis and reflects on the extent to which the original objectives were met. It also outlines possible directions for future improvements and extensions. The chapter is divided into two parts: first, a brief recap of the results and contributions, and second, a discussion of enhancements that could move the system closer to production grade maturity.

7.1 Summary of Achievements

This thesis set out to explore how open-source Iac tools can be combined to design a reproducible, modular, and secure infrastructure for SMEs. Using Proxmox, Terraform, Ansible, Docker, Traefik, Keycloak, Snipe-IT, and Vault, a working environment was successfully provisioned, secured, and validated.

The objectives of automation, modularity, and improved security were achieved. The implementation demonstrated how such an infrastructure can reduce manual effort, improve reproducibility, and provide a strong baseline for cybersecurity-oriented organizations.

While not production-grade, the setup serves as a proof of concept and a foundation for future work. The results highlight the practical value of Iac in improving both operational efficiency and security, making it a viable approach for SMEs with limited resources.

7.2 Future Work

While the current infrastructure presents a solid and automated foundation, several key enhancements would elevate it to a production-grade system, particularly in environments where uptime and resilience are critical such as in cybersecurity-focused organizations.

A major future enhancement involves implementing a high-availability (HA) cluster using Proxmox VE with multiple physical nodes. Proxmox natively supports clustering, allowing VMs to be automatically migrated or restarted on other nodes in the event of hardware failure or maintenance. This feature is particularly valuable in sectors like cybersecurity, where minimizing downtime is essential to maintaining service integrity and operational trust.

Although deploying a Proxmox VE HA setup introduces additional hardware and networking requirements, its benefits in ensuring business continuity are substantial. For SMEs in the cybersecurity field, the investment is justified by the reduced risk of operational interruption and the improved ability to perform secure updates or recovery without service disruption. In this enhanced architecture, services like Keycloak, Vault, and internal documentation systems could maintain availability even during infrastructure failures.

In addition to high availability, further improvements would include the deployment of centralized monitoring and logging. Solutions like Prometheus and Grafana for metrics, along with Loki or the ELK stack for log aggregation, would provide better observability and early detection of anomalies or breaches. These tools are vital for responding to incidents quickly and meeting audit and compliance requirements.

CI/CD practices also remain a key focus. Expanding the current use of GitLab CI to include more comprehensive automated testing, linting, and staged rollouts would improve the reliability of deployments. Secured Git workflows, integrated with tools like Vault for secret management, would reinforce the integrity of the

release pipeline.

Lastly, integrating an automation platform such as Windmill could offer long-term advantages. Windmill enables safe internal scripting and user-friendly automation of tasks, making internal tools more accessible to non-technical teams while still preserving access control and logging.

Altogether, these future enhancements particularly the adoption of Proxmox VE HA would move the infrastructure closer to enterprise-grade reliability, aligning it with the needs of cybersecurity organizations that cannot afford significant downtime or operational drift.

References

[1] M. Kandasamy and S. Kanungo, *Automate hybrid cloud infrastructure provisioning with terraform and ansible*, 2021. [Online]. Available: https://www.ibm.com/think/tutorials/automate-hybrid-cloud-infrastructure-provisioning-with-terraform-and-ansible.

[2] S. Carolan and D. Silva, *Ansible and terraform: Better together*, 2022. [Online]. Available: https://www.hashicorp.com/en/resources/ansible-terraform-better-together.

[3] M. Zapf, *Improve application security using a reverse proxy*, 2023. [Online]. Available: https://traefik.io/blog/improve-application-security-using-a-reverse-proxy/.

[4] M. McCabe, *The security benefits of infrastructure as code*, 2023. [Online]. Available: https://blog.cloudsecuritypartners.com/the-security-benefits-of-infrastructure-as-code/.

[5] F. Dinu and K. Galbraith, *The business benefits of infrastructure as code*, 2022. [Online]. Available: https://spacelift.io/blog/business-benefits-of-iac.

[6] SentinelOne, *Infrastructure as code principles: What you need to know*, 2023. [Online]. Available: https://www.sentinelone.com/cybersecurity-101/cloud-security/infrastructure-as-code-principles/.

[7] V. Krishnakumar, *Infrastructure as code: A complete guide to modular design, compliance and monitoring*, 2023. [Online]. Available: https://www.cloudoptimo.com/blog/infrastructure-as-code-a-complete-guide-to-modular-design-compliance-and-monitoring/.

[8] N. M. K. Koneru, "Infrastructure as code (iac) for enterprise applications: A comparative study of terraform and cloudformation", *American Journal of Technology*, vol. 4, no. 1, pp. 1–28, 2025. [Online]. Available: https://doi.org/10.58425/ajt.v4i1.351.

[9] I. Byzov, "Terraform vs ansible: When and how to use infrastructure tools as code", *Technologies and Engineering*, no. 6, pp. 11–17, 2025. [Online]. Available: https://doi.org/10.30857/2786-5371.2024.6.1.

[10] A. Pathak, "Automating infrastructure management: Benefits and challenges of ansible and terraform implementation across sectors", *International Journal of Computer Science and Engineering Technology*, vol. 11, no. 5, pp. 1032–1040, 2024. [Online]. Available: https://doi.org/10.32628/CSEIT241051032.

[11] A. Raghu, "Implementing hashicorp vault for secure credential management in financial services: A java-centric approach", *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 3, 2025. [Online]. Available: https://doi.org/10.22399/ijcesen.2473.

[12] J. Du and G. Nie, "Design and implementation of security reverse data proxy server based on ssl", in *Communications in Computer and Information Science (ICCIC)*, vol. 235, Springer, 2011, pp. 523–528. [Online]. Available: https://doi.org/10.1007/978-3-642-24022-5_83.

[13] L. Prates and R. Pereira, "Devsecops practices and tools", *International Journal of Information Security and Privacy*, 2024. DOI: https://doi.org/10.1007/s10207-024-00914-z.

[14] G. Vanam, "Infrastructure automation in cloud computing: A systematic review of technologies, implementation patterns, and organizational impact", *International Journal of Computer Engineering & Technology*, vol. 16, no. 1, pp. 55–69, 2025. [Online]. Available: https://doi.org/10.34218/IJCET_16_01_006.

[15] S. A. Algarni, M. R. Ikbal, R. Alroobaea, A. S. Ghiduk, and F. Nadeem, "Performance evaluation of xen, kvm, and proxmox hypervisors", *International Journal of Open Source Software and Processes*, vol. 9, no. 2, pp. 26–43, 2018. [Online]. Available: https://doi.org/10.4018/IJOSSP.2018040103.

[16] J. Aas, R. Barnes, B. Case, *et al.*, "Let's encrypt: An automated certificate authority to encrypt the entire web", in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, 2019, pp. 2473–2487. [Online]. Available: https://doi.org/10.1145/3319535.3363192.

[17] The Debian Project, *Why debian?*, https://www.debian.org/intro/why_debian, Accessed: 2025-07-15.

[18] M. Dorcey, "Internal documentation leads to better developer experiences—here's how", *OpsLevel Insights*, 2023, Accessed via web article. [Online]. Available: https://www.opslevel.com/resources/internal-docs-for-devx.

[19] O. Justin Onyarin, *A complete guide to devops best practices*, Zenodo, 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6376787.

[20] E. Özdoğan, O. Ceran, and M. T. Üstündağ, "Systematic analysis of infrastructure as code technologies", *GU Journal of Science, Part A: Engineering and Innovation*, vol. 10, no. 4, pp. 452–471, 2023. [Online]. Available: https://doi.org/10.54287/gujsa.1373305.

[21] H. Dasari, "Infrastructure as code (iac) best practices for multi-cloud deployments in enterprises", *International Journal of Networks and Security*, vol. 5, no. 1, pp. 174–186, 2025. [Online]. Available: https://doi.org/10.55640/ijns-05-01-10.

[22] A. Bertram, *Use active directory for testing environments in devops shops*, TechTarget SearchITOperations, Accessed: 2025-07-21, 2018. [Online]. Available: https://www.techtarget.com/searchitoperations/tip/Use-Active-Directory-for-testing-environments-in-DevOps-shops.

[23] S. Hatfield, *Creating a new active directory forest with ansible*, https://madlabber.wordpress.com/2019/09/08/creating-a-new-active-directory-forest-with-ansible/, Accessed: 2025-07-21, 2019.

[24] B. I. Mokhtar, A. D. Jurcut, M. S. ElSayed, and andMarianne A. Azer., "Active directory attacks—steps, types, and signatures", *Electronics*, vol. 11, no. 16, p. 2629, 2022. [Online]. Available: https://doi.org/10.3390/electronics11162629.

[25] D. Fett, R. Küsters, and G. Schmitz, *The web sso standard openid connect: In-depth formal security analysis and security guidelines*, Accessed: 2025-07-21, 2017. [Online]. Available: https://doi.org/10.1109/CSF.2017.20.

[26] W. Li, C. J. Mitchell, and T. Chen, *Mitigating csrf attacks on oauth 2.0 and openid connect*, Accessed: 2025-07-21, 2018. [Online]. Available: https://doi.org/10.1109/PST.2018.8514180.

[27] D. D. and N. G. Cholli, "A review on identity and access management server (keycloak)", *International Journal of Electrical and Power Engineering*, vol. 14, no. 2, pp. 17–22, 2020. [Online]. Available: https://doi.org/10.4018/IJSPPC.2020070104.

[28] S. K. Basak, L. Neil, B. Reaves, and L. Williams, *What are the practices for secret management in software artifacts?*, arXiv preprint arXiv:2208.11280, Accessed: 2025-07-21, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2208.11280.

[29] G. Mulpuri, "Integration of security tools like vault and secrets management in ci/cd pipelines", *International Journal of Science and Research*, vol. 10, no. 9, 2021. [Online]. Available: https://dx.doi.org/10.21275/SR24402110508.

[30] R. C. andDoron Pinhas, "Security guidelines for storage infrastructure", U.S. Department of Commerce, Tech. Rep. NIST SP 800-209, 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-209.

[31] G. Thiyagarajan and P. Nayak, "Docker under siege: Securing containers in the modern era", *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 11, no. 1, pp. 3674–3719, 2025. [Online]. Available: https://doi.org/10.32628/CSEIT25112773.

[32] H. A. Al-Ofeishat and R. Alshorman, "Build a secure network using segmentation and micro-segmentation techniques", *International Journal of Computing and Digital Systems*, vol. 16, no. 1, 2024. [Online]. Available: https://doi.org/10.12785/ijcds/1601111.

[33] R. C. Thota, "End-to-end automation of multi-cloud deployments using terraform, ansible, and kubernetes", *International Journal of Innovative Research in Engineering Multidisciplinary Physical Sciences*, vol. 8, no. 4, pp. 1–10, 2020. [Online]. Available: https://doi.org/10.37082/IJIRMPS.v8.i4.232184.

[34] A. Pathak, "Automating infrastructure management: Benefits and challenges of ansible and terraform implementation across sectors", *International Journal of Scientific Research in Computer Science, Engineering and Informa-*

tion Technology, vol. 10, no. 5, pp. 381–394, 2024. [Online]. Available: https://doi.org/10.32628/CSEIT241051032.

[35] C. Diekmann, J. Naab, and G. C. Andreas Korsten, "Agile network access control in the container age", *arXiv preprint arXiv:1903.00720*, 2019. [Online]. Available: https://arxiv.org/abs/1903.00720.

[36] I. S. Arnanda Nuryasa, "Implementasi traefik sebagai reverse proxy dengan prinsip zero trust security", *Jutisi: Jurnal Ilmiah Teknik Informatika dan Sistem Informasi*, vol. 13, no. 1, pp. 107–115, 2024. [Online]. Available: https://doi.org/10.35889/jutisi.v13i1.1704.

[37] F. Wilkens, S. Haas, J. Amann, and M. Fischer, "Passive, transparent, and selective tls decryption for network security monitoring", *arXiv preprint*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2104.09828.

[38] A. Verdet, M. Hamdaqa, L. D. Silva, and F. Khomh, "Exploring security practices in infrastructure as code: An empirical study", *arXiv preprint*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.03952.

[39] N. M. K. Koneru, "Infrastructure as code (iac) for enterprise applications: A comparative study of terraform and cloudformation", *American Journal of Technology*, vol. 4, no. 1, pp. 1–28, 2025. [Online]. Available: https://doi.org/10.58425/ajt.v4i1.351.

[40] P. R. R. Konala, V. Kumar, D. Bainbridge, and J. Haseeb, "A framework for measuring the quality of infrastructure-as-code scripts", *arXiv preprint*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.03127.