DUALE HOCHSCHULE BADEN-WÜRTTEMBERG MANNHEIM
Business Information Science – International Management for Business and Information Technology (IMBIT)

AUTOMATING THE IT LANDSCAPE: A PRACTICAL IMPLEMENTATION OF INFRASTRUCTURE AS CODE USING ANSIBLE AND TERRAFORM

Bachelor Thesis

submitted by Agostino Fabio Sauna

Student ID, Class: [redacted]
Academic Supervisor:
Company Supervisor: [redacted]
Company: Cosmo Consult SI GmbH
Location, Date: Mannheim, 08.04.2024

Declaration of academic integrity

I hereby declare, that:

- I have written the thesis “Automating the IT Landscape: A Practical Implementation of Infrastructure as Code Using Ansible and Terraform” on my own, i.e., without the help of a third party, and that the work is my own.
- I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document.
- I have not submitted my thesis to any other university or authority.
- I have not published my thesis in the past.
- I am aware that a dishonest declaration will entail legal consequences.

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: „ Automating the IT Landscape: A Practical Implementation of Infrastructure as Code Using Ansible and Terraform“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

_____ _____

Ort, Datum Unterschrift

Abstract (English)

Cloud computing has revolutionized the field of IT, enabling businesses to scale their resources with agility. However, traditional administration of IT environments often results in inefficiencies when it comes to scaling. Infrastructure as Code (IaC) emerges as a solution to allow automated infrastructure provisioning and configuration.

This study presents an exemplary IaC deployment that addresses the needs of small and medium-sized enterprises (SMEs) by providing a comprehensive and open-source artifact using Ansible and Terraform. The article presents a case study that uses Design Science Research methodology to develop a practical IaC model. The artifact demonstrates a templating mechanism for Docker services, centralized URL management for service configuration and monitoring, and the management of a hybrid cloud infrastructure.

The contribution of this research is a detailed example that provides practical insights for SMEs looking to adopt IaC. Despite limitations in server provisioning, this study represents a significant step towards reducing the implementation barriers for IaC. It is recommended that future research extend this work by developing more open-source solutions.

Abstract (German)

Cloud Computing hat den IT-Bereich revolutioniert und ermöglicht es Unternehmen, ihre Ressourcen flexibel zu skalieren. Die herkömmliche Administration von IT-Umgebungen führt jedoch oft zu Ineffizienzen bei der Skalierung. Infrastructure as Code (IaC) ist eine Lösung, die eine automatisierte Infrastrukturbereitstellung und -konfiguration ermöglicht.

In dieser Studie wird eine beispielhafte IaC-Lösung vorgestellt, die auf die Bedürfnisse kleiner und mittlerer Unternehmen (KMU) ausgerichtet ist, indem ein umfassendes und quelloffenes Artefakt unter Verwendung von Ansible und Terraform bereitgestellt wird. Die vorliegende Bachelorarbeit stellt eine Fallstudie vor, die die Methodik der Design Science Research zur Entwicklung eines praktischen IaC-Modells nutzt. Das Artefakt demonstriert einen Templating-Mechanismus für Docker-Dienste, ein zentralisiertes URL-Management für die Dienstkonfiguration und -überwachung sowie die Verwaltung einer hybriden Cloud-Infrastruktur.

Der Beitrag dieser Forschung ist ein detailliertes Beispiel, das praktische Einblicke für KMUs bietet, die IaC einführen möchten. Trotz der Einschränkungen bei der Serverbereitstellung stellt diese Studie einen wichtigen Schritt zur Verringerung der Implementierungsbarrieren für IaC dar. Basierend auf den Resulataten der vorliegenden Arbeit wird empfohlen, dass künftige Forschungen diese Arbeit durch die Entwicklung weiterer quelloffene Lösungen erweitern.

I. Table of Content

II. List of Figures

III. List of Tables

IV. List of Abbreviations

Abbreviation	Definition
API	Application Programming Interface
BI	Business Intelligence
CD	Continuous Delivery or Continuous Deployment
CI	Continuous Integration
CM	Continuous Monitoring
CMS	Content Management System
CPU	Central Processing Unit
CRM	Customer Relationship Management
DHCP	Dynamic Host Configuration Protocol
DMS	Document Management System
DMZ	Demilitarized Zone
DNS	Domain Name System
DSR	Design Science Research
DSRM	Design Science Research Methodology
ERP	Enterprise Resource Planning
GPO	Group Policy Object
HDD	Hard Disk Drive
HRM	Human Resources Management
HSTS	HTTP Strict Transport Security
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAM	Identity and Access Management
IDE	Integrated Development Environment
IoT	Internet of Things
IS	Information Systems
IT	Information Technology
KVM	Kernel-based Virtual Machine
LMS	Learning Management System
LTS	Long-Term Support
LXC	Linux Container
NAS	Network-Attached Storage
NGFW	Next-Generation Firewall
OCR	Optical Character Recognition
OS	Operating System
PaaS	Platform as a Service
RAM	Random Access Memory
SCCM	System Center Configuration Manager

SCM	Supply Chain Management
SIP	Session Initiation Protocol
SMART	Specific, Measurable, Achievable, Relevant, Time-bound
SME	Small and Medium-Sized Enterprise
SSD	Solid-State Drive
SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single Sign-On
SaaS	Software as a Service
UPS	Uninterruptible Power Supply
VCS	Version Control Software
VM	Virtual Machine
VPN	Virtual Private Network
VPS	Virtual Private Server
VScode	Visual Studio Code
WSL	Windows Subsystem for Linux
WSUS	Windows Server Update Services
YAML	Yet Another Markup Language

1. Introduction

The traditional IT environment is typically characterized by IT services installed directly on physical machines located on the company's premises, providing overall value to the business operations. In these environments, software is usually installed on virtual machines that operate on hypervisors such as VMware ESXi, Microsoft Hyper-V, or KVM (Erl et al., 2013, pp. 56–57).

However, with the advent of the cloud era, this landscape has undergone a radical transformation. Readily available computing resources in the cloud have transformed the processes of deploying, managing, and configuring IT infrastructure and services (Filkins, 2018, p. 5). Unlike traditional IT environments, where scaling infrastructure to meet fluctuating demands is a slower and more static process, organizations in the cloud era can utilize elastic and self-service computing, adding, or removing resources swiftly to match their needs. This feature enables organizations to access large amounts of computing power quickly, while only paying for the resources they consume (Abts and Mülder, 2017, p. 157; Arul Elumalai et al., 2018, p. 9).

Despite these advancements, the administration of both traditional and cloud environments still largely involves many manual tasks, which can present significant challenges. As organizations grow and their systems become more complex, manual system administration can become inefficient and lead to bottlenecks and suboptimal performance (Geerling, 2020, pp. VII–VIII; Wurster et al., 2020, p. 63).

Moreover, the structure of the applications themselves can exacerbate these challenges. IT services have traditionally been configured manually, leading to the creation of monolithic applications. Such applications, being a fusion of various components into a single coherent program, are highly interdependent. As a result, even minor modifications can impact the entire system (Spinellis, 2012, p. 86). Managing and replicating a fully configured machine with these monolithic applications can be a daunting task (Been et al., 2022, pp. 7–8; John and Douglas, 2019, p. 2). Issues such as the absence of current documentation worsen the situation, alongside configuration drift—which refers to unsanctioned or incorrect changes that create discrepancies over time within IT environments (Geerling, 2020, pp. 1–2; Morris, 2021, pp. 17–18).

According to recent research, human error is a major contributor to project failures and accounts for a significant 74% of security breaches (C. David Hylender et al., 2023, p. 8; Humble and Farley, 2010, p. 279) This serves to underscore the substantial risks of manual IT processes, which not only slow down business innovation but can lead to ineffective resource use and a lack of the agility and promptness which modern enterprises require (Artac et al., 2017, p. 497; Guerriero et al., 2019, p. 581).

To address these challenges, Infrastructure as Code (IaC), a DevOps practice, has emerged as a viable solution. This technique utilizes code to automatically manage and provision infrastructure, reducing human error and increasing agility and effectiveness.

The roots of IaC can be traced back to the creation of CFEngine in 1993 (Burgess, 1995) and was later advanced by tools such as Puppet in 2005 (Kanies, 2006) and Chef in 2009 (Robbins, 2009). These tools laid the groundwork for infrastructure management that is both automated and codified. While DevOps, introduced by Patrick Debois in 2009, encompasses a broader range of coding practices, it has significantly reinforced the role of IaC (Hüttermann, 2012, p. 5).

The advent of cloud computing has played a significant role in the increasing popularity of Infrastructure as Code (Been et al., 2022, p. 4; Forrester Consulting, 2022, p. 12). This surge in interest is also evidenced by the upward trends in Google searches for Cloud Computing and DevOps, as depicted in Figure 1. Corroborating this trend, according to Fortune Business Insights, the market outlook for IaC technologies is impressive, with projections indicating a growth from $908.7 million in 2023 to $3,304.9 million by 2030 (Fortune Business Insights, 2023, p. 1).

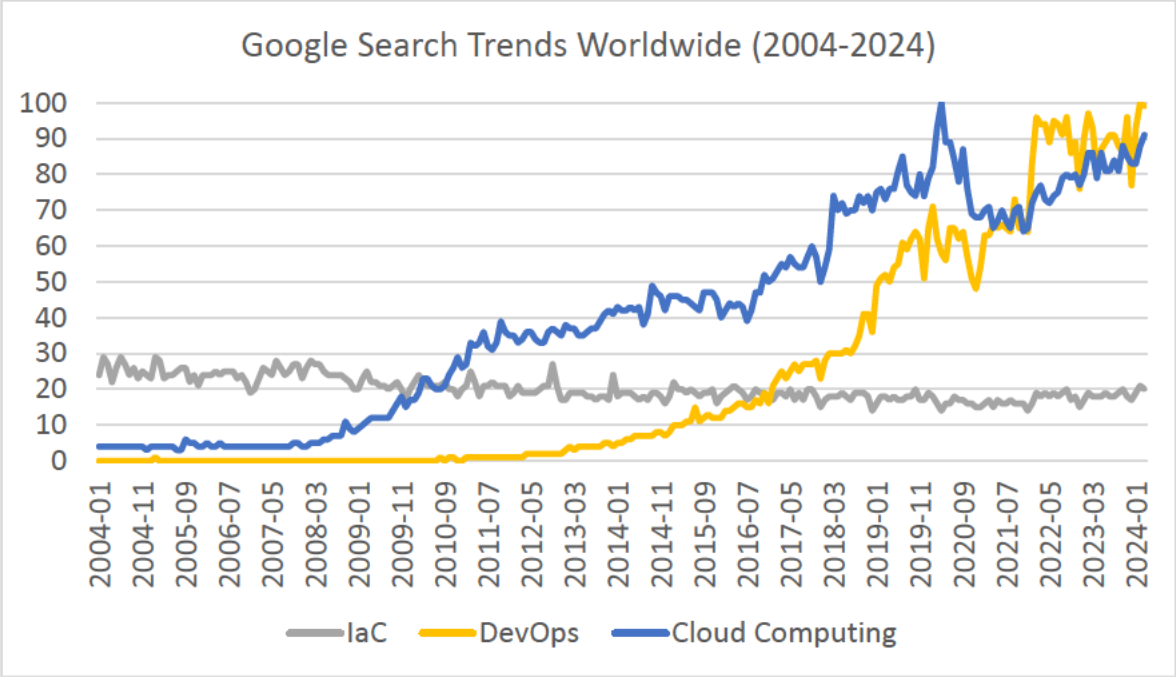


Figure 1: Google Search Trends Worldwide (2004-2024) (Google Trends, 2024).

1.1 Current Research and Problem Scope

Several studies have explored the field of IaC, providing valuable insights into its advantages and challenges (Guerriero et al., 2019; Rajapakse et al., 2022; Riungu-Kalliosaari et al., 2016). Empirical investigations, including those conducted by Kropp et al. (2016) and Limoncelli (2018), have further confirmed these findings in practical settings. The social

dynamics within organizations that adopt IaC have been analyzed, revealing a shift in collaborative practices (Kropp et al., 2016; Limoncelli, 2018). Efforts have been made to distill collective wisdom into best practices (Kumara et al., 2021; Rahman et al., 2020). Additionally, other research has addressed security concerns (Chen et al., 2018; Rahman et al., 2021; Rahman, Parnin, Williams, 2019; Schwarz et al., 2018). Scholars such as John and Douglas (2019) and Tomas et al. (2019) have further evaluated the overall practicability of IaC (John and Douglas, 2019; Tomas et al., 2019).

While there is a significant amount of literature on the subject, it lacks concrete examples, such as actual source code or detailed implementation narratives. Guerriero et al. (2019) and Rahman, Mahdavi-Hezaveh, Williams (2019) have identified a research gap in this area, which still holds true today (Guerriero et al., 2019, pp. 586–587; Rahman, Mahdavi-Hezaveh, Williams, 2019, pp. 36–37).
While it is understandable that IaC setups may be proprietary and infrastructural configurations may be confidential, the lack of information can pose a significant challenge for small and medium-sized enterprises (SMEs) who are interested in adopting and understanding IaC. SMEs may face challenges in terms of resources for innovation, which can impact their ability to transition to Infrastructure as Code effectively. Therefore, it is important to provide clear and actionable information to support informed decision-making in this area (Alshamaila et al., 2013, pp. 250–251).

Based on these gaps in the literature, the following research question is proposed:

i. What would an exemplary Infrastructure as Code deployment for SMEs look like?

1.2 Methodology

This bachelor thesis employs Design Science Research (DSR) as its methodology, with a focus on developing a solution to bridge the knowledge gap in migrating IT infrastructure to Infrastructure as Code for small to medium-sized enterprises.
The research employs Peffers et al.'s six-step DSR methodology, beginning with problem identification through literature review and empirical evidence via a survey, and progressing towards defining objectives for the solution based on a descriptive case study of a Homelab environment. The design and development step utilizes literature research to inform the creation of the IaC environment, which then undergoes a descriptive demonstration to showcase its practical application. The evaluation of the artifact in bridging the research gap is assessed by measuring its alignment with predefined objectives. These objectives are established based on SMART criteria and are evaluated through a combination of monitoring and manual verification specific to the case study context. The research concludes with the communication step, where the findings and complete source code are publicly disseminated

via GitHub and other relevant platforms, with the aim of sharing the acquired knowledge with a wider audience.
This structured approach promotes a systematic and transparent research process that aligns with the standards for undergraduate theses and offers practical contributions to address the research gap.

1.3 Objectives and Scope

The aim of this research is to contribute to the existing body of knowledge and narrow the significant knowledge gap in the field. The primary intention is to introduce an artifact that serves as a pioneering example, while acknowledging that a single study cannot completely bridge this gap.
The development of this artifact focuses on its relevance and practicality for SMEs within the modern IT landscape. To guarantee relevance and practicality, it is essential to adopt a methodology that aligns with the research objectives and addresses the specific needs of this sector. It is important to gather insights to determine what is considered crucial within the domain of Infrastructure as Code (IaC). These identified needs must then be practically developed into the artifact, ensuring that it provides SMEs with relevant source code. To ensure a complete understanding and possible replication of the source code, it is critical to establish a theoretical framework that explains the basic principles of Infrastructure as Code. This should be accompanied by comprehensive explanations of the source code concepts.
To effectively engage with the content, readers are expected to have a foundational understanding of systems engineering, basic programming, Docker and networking. A comprehensive explanation of these disciplines is beyond the intended scope of this research. For those seeking to build or refresh their knowledge in these areas, Kersken's seminal work "IT-Handbuch für Fachinformatiker" by Kersken (2019) (Kersken, 2019) is recommended.
Additionally, this thesis cannot provide an exhaustive explanation of the coding involved. A comprehensive guide to tutorials and comparisons of various IaC tools is beyond the scope of this study. Readers interested in a detailed examination of IaC tools, and their applications are advised to consult specialized resources such as "Ansible for DevOps" by Geerling (2020), "Service-Oriented Infrastructure: On-Premise and in the Cloud" by Hochstein and Moser (2017), and "Terraform: Up & Running" (2022) by Morris.

1.4 Outline

Chapter 2 discusses DevOps and Infrastructure as Code, providing a comprehensive understanding of both topics and their interrelated aspects, laying a solid foundation for the subsequent sections. In Chapter 3, the methodology employed for the research is explained, including the approach to data collection, analysis methods, and the overall research design.

Chapter 4 presents the survey findings, confirming the gap in the literature and providing guidance for creating an artifact relevant to SMEs. Chapter 5 selects, validates, and describes the case study environment from which the practical-driven objectives for the artifact will be derived. Chapter 6, the core of the thesis, delves into the design and code specifics of the artifact, elucidating the source code and its function in detail.
Chapter 7 then outlines the research findings. Subsequently, Chapter 8 assesses and discusses these findings in the context of the research objectives, delivering an in-depth analysis of the results.
Concluding the thesis, Chapter 9 summarizes the key findings of the research and suggests avenues for future research.

2. Background of Infrastructure as Code

This chapter explores the core principles and practices of IaC, a modern approach rooted in the DevOps movement that is transforming the IT landscape. This fundamental knowledge is crucial, as it forms the basis for comprehending the development of the artifact discussed in subsequent chapters. Due to space constraints and considering that only selected IaC can be used for developing the artifact, this chapter will only focus on general concepts and not discuss details about IaC tools.

2.1 DevOps

DevOps is a philosophy that has been instrumental in transforming software development and delivery, promoting collaboration, and improving performance in the IT sector. As seen in Figure 2, it combines the **dev**elopment and **op**erations teams, aiming to reconcile the often-conflicting goals between these two teams within organizations (Bass et al., 2015, pp. 21–22; Hüttermann, 2012, p. 4). The essence of DevOps lies in a set of practices, one of which is Infrastructure as Code, that resolve the conflict by guiding the process from code development to deployment (Been et al., 2022, pp. 5–6; Riungu-Kalliosaari et al., 2016, p. 590).

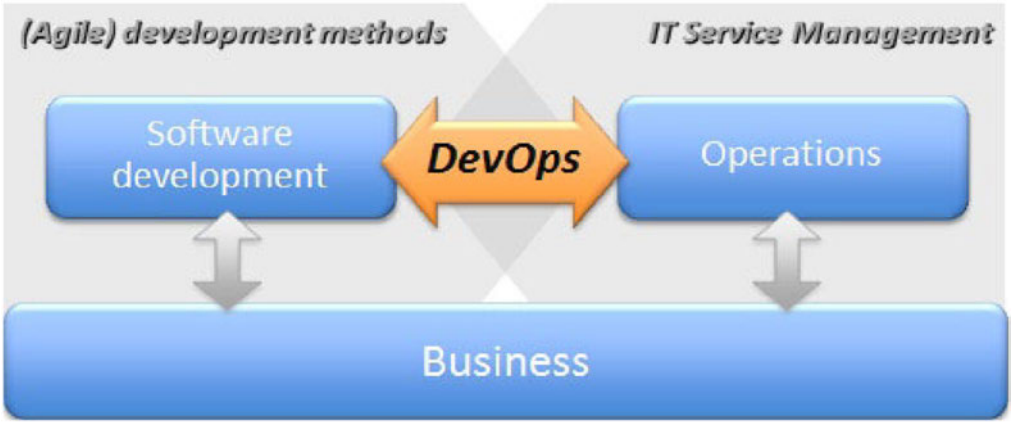


Figure 2: DevOps Linking Both Teams (Hüttermann, 2012, p. 25).

2.1.1 The Silo Problem

As depicted in Figure 3, the conflicting goals and incentives of the disciplines development and operation often lead to organizational inefficiencies (Artac et al., 2017, p. 497). Development teams prioritize innovation and rapid feature delivery, while operations teams prioritize system stability and reliability. These differing objectives can lead to a culture of conflict, as development aims for constant change while operations strive to maintain the status quo (Hüttermann, 2012, pp. 5–7).

This contrast can be viewed through the lens of their respective incentives: developers are incentivized to successfully deploy their code into production, following the adage 'release

early, release often' to iterate and improve their work. Conversely, operations teams are incentivized to minimize downtime, following the principle of “if it ain't broke, don't fix it” (Bass et al., 2015, p. 44). Operations must carefully analyze potential causes of system instability and be cautious about introducing changes that could disrupt the smooth functioning of production environments (Hüttermann, 2012, p. 19).

The lack of collaboration and shared objectives between these two departments can result in a siloed approach to software delivery. This approach can hamper the efficiency and quality of the product. It not only slows down the time to market but can also lead to inferior software quality and an increased number of system outages (F. Beetz and Simon Harrer, 2022, p. 71).

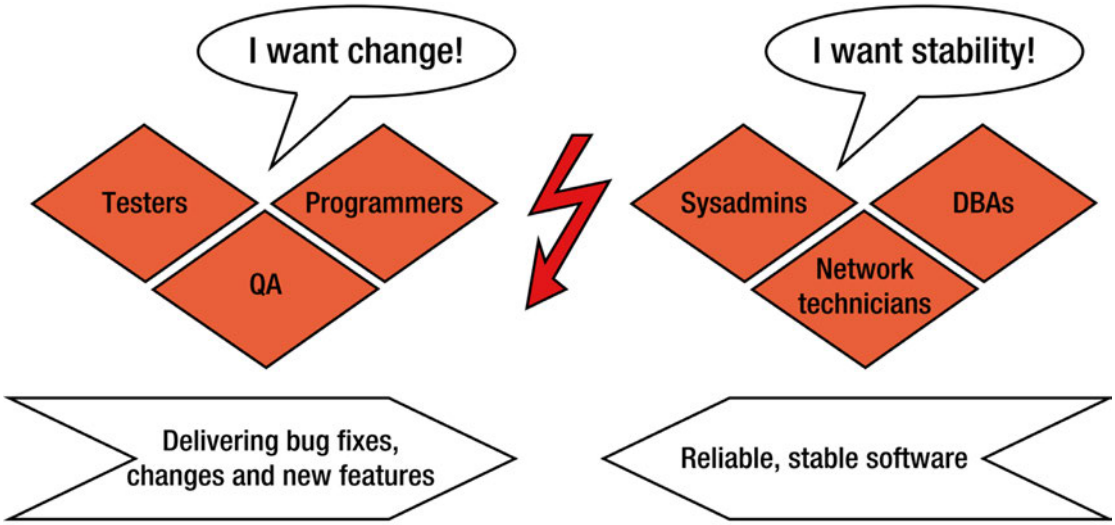


Figure 3: Example conflict across both silos (Hüttermann, 2012, p. 20).

2.1.2 DevOps Practices

DevOps aims to create faster feedback loops and minimize the risks associated with software development by adopting principles from Agile and lean production (F. Beetz and Simon Harrer, 2022, p. 71).

It promotes a collaborative environment where development and operations teams work closely together. According to Lwakatare et al. (2019, p. 2) DevOps is not just a set of practices or a technical strategy; it is a cultural movement that advocates for a mindset shift. This partnership allows for shared responsibility throughout the software lifecycle, including development, testing, deployment, and infrastructure management (Lwakatare et al., 2019, p. 218).

DevOps integrates agile principles, automates deployment processes, and employs Infrastructure as Code (IaC). These practices can facilitate a smooth workflow, enabling the rapid and dependable delivery of software improvements to users (Geerling, 2020, p. viii; Been et al., 2022, pp. 5–6). Furthermore, the DevOps approach promotes a continuous

feedback loop, where insights obtained from production are incorporated into development, creating an environment of ongoing learning and improvement (Hüttermann, 2012, p. 4).
Although DevOps is a widely used term, it has been subject to debate due to its broad and evolving nature. Scholars have noted the importance of clarity and empirical validation in understanding the impact of DevOps practices on software release cycles and quality (F. Beetz and Simon Harrer, 2022, p. 71).
However, practitioners widely acknowledge that collaboration and communication between development and operations are fundamental aspects of the DevOps ethos (Lwakatare et al., 2019, p. 224).

2.1.2.1 Version Control

Version control is a crucial aspect of modern software development. It serves as a central repository that stores all code and meticulously records changes done to them (Morris, 2021, p. 37). Originally designed only for source code management, Version Control Software (VCS) has evolved to handle various file types, including build and deployment scripts, documentation, configuration files, and even tools and libraries essential to development (Humble and Farley, 2010, p. 33). While most version control systems share similar features for unified change management, this section will solely focus on the terminology and main features of the popular VCS Git, as it is used in the case study (Beetz and Harrer, 2022).
In Git, all modifications, no matter how small, are meticulously tracked and evaluated through a consistent ticketing system, which facilitates a comprehensive history of changes. This record includes information about who made each change and the reasons behind them, simplifying the process of diagnosing and addressing issues (Morris, 2021, pp. 37-38). This detailed documentation not only ensures accountability and facilitates audits but also supports the system's ability to return to a previous stable state if necessary. These rollbacks are particularly critical in complex environments where new changes may inadvertently introduce issues (Morris, 2021, pp. 37-38 Klein & Reynolds, 2019).
Furthermore, the traceability of changes enhances team situational awareness, empowering members to quickly understand the implications of new changes and identify potential issues that could surface (Morris, 2021, pp. 37-38). This level of insight proves invaluable when integrating new team members, as they are granted access to the entire project history and all necessary artifacts, guaranteeing a seamless integration into the project (Humble and Farley, 2010, p. 33).
However, when it comes to security, it is crucial to avoid storing sensitive information, such as passwords, in an unencrypted form within these systems to prevent data breaches. To ensure security and compliance, it is recommended to either use dedicated solutions to manage such secrets or keep such files outside of version control. This is especially important if the repositories are publicly accessible (Morris, 2021, pp. 37-38).

The case study defines variables containing sensitive information in multiple central secrets files that are excluded from version control. These files are omitted from version control by listing their relative paths in the *.gitignore* file.
Applying version control to Infrastructure as Code provides the same benefits as previously mentioned (Bass et al., 2015, p. 238; Artac et al., 2017, p. 497). This practice ensures version-controlled management of the infrastructure configuration files, establishing a single source of truth for the system's state, promoting consistency and collaboration among team members (Artac et al., 2017, p. 497). Furthermore, version control systems aid in the precise replication and management of various IT environments, leading to more dependable and controlled testing procedures (Klein and Reynolds, 2019).

Version control systems not only facilitate change documentation but also trigger automated actions and processes post-commit. These functionalities are essential for the practice continuous integration and deployment, which will be later discussed.

2.1.2.2 Evolution from Agile Development

Agile methodologies aim to break down the compartmentalized structure of traditional project phases, where distinct groups such as programmers, testers, and quality assurance operate in isolation (Hüttermann, 2012, pp. 8–9). Instead, Agile fosters an integrated, cross-functional team dynamic, promoting continuous collaboration. This shift has led Agile to surpass the success rates of traditional models like Waterfall by enabling more effective responses to changes and a higher likelihood of delivering functional software (Williams and Vouk, 2009).
The Agile approach emphasizes customer satisfaction by providing valuable software early and continuously through iterative cycles of planning, execution, and evaluation. This results in a series of progressive releases that incrementally advance towards the final product (Williams and Vouk, 2009). The efficacy of Agile is demonstrated by its widespread adoption by tech giants such as Facebook, GitHub, and Netflix (Rahman et al., 2015, p. 1).

DevOps is based on a collaborative ethos that promotes shared responsibility and continuous improvement. It is an evolution of Agile methodologies, extending Agile principles to system administration and IT operations (Bass et al., 2015, pp. 32–33; Hüttermann, 2012, pp. 8–9, 2012, p. xvi). Therefore, the development of Infrastructure as Code should adhere to Agile methods, ensuring that infrastructure management benefits from the same iterative and responsive approach as application development.
Integrating DevOps practices into the Agile framework not only complements and strengthens the existing roles within Agile teams, but it's also a vital component for the successful adoption of DevOps principles (Lwakatare et al., 2016, p. 223). By applying

Agile's emphasis on adaptability and rapid incorporation of feedback, organizations can smoothly transition into a DevOps model, which further enhances the efficiency of the development cycle (Lwakatare et al., 2019, p. 218).

The adoption of DevOps practices within Agile frameworks catalyzes a synergy that significantly accelerates delivery times, elevates software quality, and heightens customer satisfaction. DevOps advocates for effective communication and a strong commitment to these goals, rather than relying solely on tooling (Hüttermann, 2012, pp. 8–9). This approach ensures the alignment of technological and human components of software development, driving innovation and efficiency in Agile-DevOps strategies (Lwakatare et al., 2019, p. 218).

Key Agile practices such as continuous integration, deployment, and delivery, integral to the Agile-DevOps convergence, will be explored in detail in the following section.

2.1.2.3 Automated Build and Deployment Pipelines

Traditional development models often encounter "integration hell," a term used to describe the difficulty of merging weeks or months of disparate coding efforts, inevitably resulting in numerous conflicts and bugs that are difficult to resolve. Continuous integration (CI) stands in stark contrast to this problematic scenario. By integrating code changes into a shared repository multiple times a day, CI promotes a culture of frequent, automated builds and testing (Chen, 2017). This approach helps maintain a stable codebase, which is essential in a market characterized by dynamic requirements (Humble and Farley, 2010). The practice of iterative integration supports rapid bug detection, making the development process more reliable and efficient. As a result, developers can focus on creating new features rather than fixing bugs by automating repetitive tasks, resulting in more effective time management, optimal resource utilization, accelerated development cycle, and continuous delivery of value to users (Satvik Garg et al., 2021).

Continuous delivery (CD) enhances this by ensuring, through automated testing, that any code changes that pass through the CI pipeline are automatically processed to the point where they can be considered ready for live production. This state of readiness means that the code is not only functional, but also fully packaged and vetted through rigorous quality checks, requiring only a final sign-off to move into production.

Building on this foundation, Continuous Deployment (also known as CD) eliminates this final human-triggered step by automatically pushing changes into production if they meet pre-defined quality standards. Continuous deployment is characterized by a workflow in which incremental and frequent changes are deployed immediately. The adoption of continuous delivery and deployment minimizes the risks associated with big-bang releases through small, manageable updates, and facilitates the rapid detection and correction of problems to maintain production stability (Humble and Farley, 2010).

In the complex ecosystems of microservices architectures and cloud computing environments, the deployment flexibility provided by continuous deployment becomes a critical enabler. Automation, which is moving from a luxury to a necessity, underpins this methodology (Balalaie, Heydarnoori, et al., 2016; Chen, 2017). Infrastructure that can be automatically provisioned and managed, such as Infrastructure as a Service (IaaS), complements Continuous Delivery by providing the means to fluidly move software from development to production (Chen, 2017).

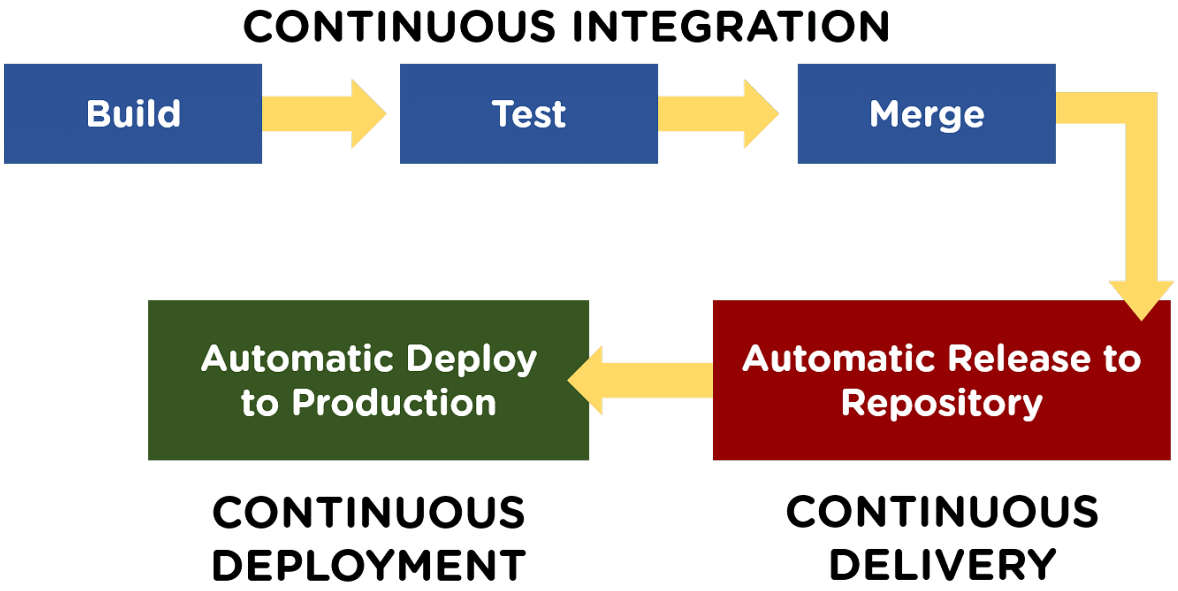


Figure 4: Complete CI/CD Pipeline (Gaba, 2020).

However, the path to full automation of the CI/CD pipeline comes with its own set of hurdles. The evolution to an automated CI/CD framework is not an instantaneous transformation, but a gradual shift that requires broad organizational buy-in (Labouardy, 2021). The incremental nature of this shift allows organizations to expand their deployment strategies and adapt to the ever-changing terrain of CI/CD advancements.

It's important to recognize that CI/CD is not a silver bullet that applies to all organizational scenarios. As Humble and Farley (2010) note, certain organizations, especially those that are highly regulated or have complex support mechanisms, may find CI/CD adoption to be a formidable challenge. The demands of regulatory compliance can sometimes conflict with the flexible methods of CI/CD, requiring a more deliberate and measured release strategy.

In addition, the pursuit of end-to-end automation may not always align with business objectives, especially when considering the law of diminishing returns in terms of cost. The upfront cost of automation tools and their ongoing refinement can represent a significant financial commitment. CI/CD adoption requires specialized tools and systems that have traditionally been managed by operations teams (Myrbakken & Palacios, 2017). With increased automation, DevOps practices require professionals to have advanced

programming skills to effectively manage and maintain these systems (Humble and Farley, 2010; Labouardy, 2021; Yuen, 2021).

More fundamentally, the adoption of CI/CD practices ushers in a new age of speed and adaptability, allowing teams to quickly identify and address any problems while improving the quality of their final products and fostering innovation within their organizations (Labouardy, 2021, pp. 46-51). With the continued growth of cloud technologies, the importance of seamlessly weaving CI/CD into DevOps methodologies has become increasingly apparent. The integration of mechanisms such as CI/CD pipelines has grown essential to the success of cloud-focused strategies (Forrester Consulting, 2022, p. 7) (Humble and Farley, 2010, pp. 55-56; Labouardy, 2021, pp. 44-46; Yuen, 2021, p. 6).
While it is true that the initial investment and effort required to automate infrastructure is significant, the strategic payoff of such an endeavor is clear. For organizations seeking to maintain competitive advantage and agility in an ever-changing marketplace, establishing an automated infrastructure that supports rapid deployment cycles and CI/CD effectiveness is the foundation for lasting success.

2.2 Infrastructure as Code

Infrastructure as Code is a practice that uses code to manage IT infrastructure, overlapping, as seen in Figure 5, with principles from DevOps (Morris, 2021, p. 4). It employs a variety of tools to navigate different stages of the infrastructure lifecycle, such as server templating, provisioning, orchestration, scripting, and configuration management (A. Rahman et al., 2020, p. 752; Brikman, 2022, p. 24).
Provisioning refers to the process of setting up and preparing the (virtual) infrastructure components, such as servers, databases, and storage into their default post installation state (Hüttermann, 2012, pp. 135–136).
Configuration management instead is the process of adapting and maintaining computer systems, servers, and software in a desired state. It involves ensuring that all systems are configured to predetermined specifications through code, which can automatically apply these settings across diverse environments (Been et al., 2022, pp. 11–15; Brikman, 2022, pp. 26–28; Hochstein and Moser, 2017, pp. 2–8).
IaC relies on several key practices that are essential for its successful implementation. The concept of IaC is based on the idea that all infrastructure elements and modifications should be managed through code changes that are systematically tested and deployed using automated processes to prevent configuration drift. When the actual state of the infrastructure differs from the state defined by the code, this is called “configuration drift”. Usually this happens when someone makes manual changes to the infrastructure outside the code (Morris, 2021, p. 19, 2021, pp. 17–18; Yuen, 2021, p. 16). In IaC this is highly

discouraged as it aims to prevent inconsistencies that would be caused by such manual changes. Instead, every change to the infrastructure is supposed to be solely executed through code (Been et al., 2022, pp. 8–11).

It is important to adhere to continuous integration and delivery for every piece of infrastructure. Components should be declared small, simple, idempotent, and independent from each other so that they can be altered without impacting the broader system (Morris, 2021, pp. 9–11). Idempotent code ensures that the consistent execution of the same code, with the certainty that the result will not change after its initial application. For instance, instead of repeatedly adding a user to a system file, which would create redundancy, an idempotent operation ensures that there is only one valid instance of the user entry, regardless of how many times it is added (Morris, 2021, p. 42; Yuen, 2021, p. 15, 2021, p. 19, 2021, p. 15, 2021, p. 19).

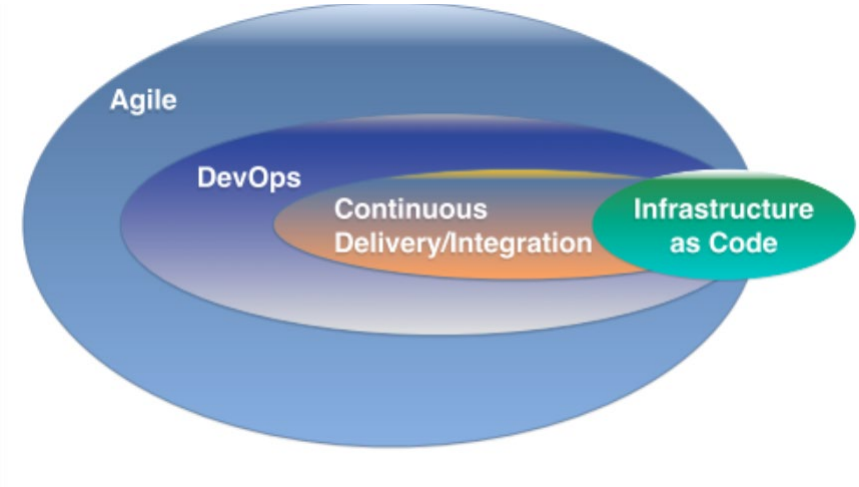


Figure 5: IaC Overlapping With Other Disciplines (John and Douglas, 2019, p. 3).

2.2.1 Declarative and Imperative Coding

When discussing IaC, there is a debate between declarative and imperative coding paradigms. Declarative code specifies the intended outcome without detailing the execution process, while imperative code outlines the exact sequence of actions to achieve a result (Yuen, 2021, pp. 13–14).

Declarative infrastructure languages focus on defining the desired state of an infrastructure without delving into the mechanics of its realization. This approach enables cleaner code and a more straightforward expression of intent. It presents a form that aligns more with configuration than programming. For example, modifying server attributes such as RAM involves simply editing the declarative code file and reapplying the tool. The tool manages the task of bringing the infrastructure into the desired state (Brikman, 2022, pp. 39–47).

On the other hand, imperative languages excel in dynamic environments where outcomes depend on certain conditions. For instance, if various sets of resources need to be created,

such as VLANs, an imperative approach would enable to use conditions and loops to handle these variations (Morris, 2021, pp. 39–44).

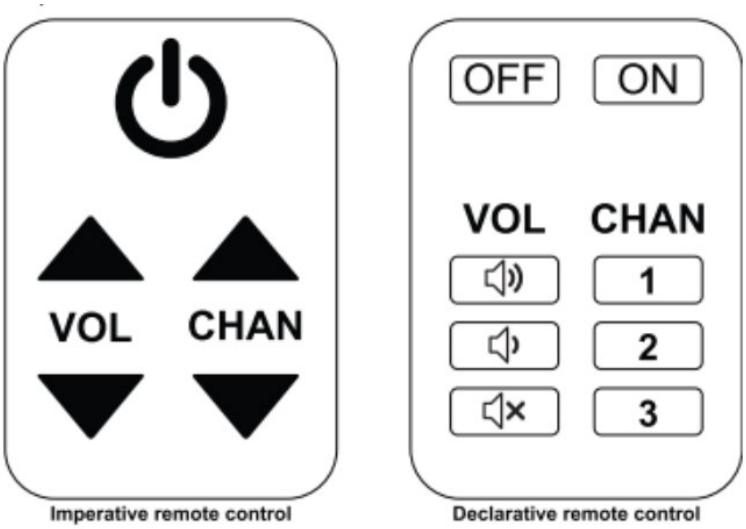


Figure 6: Two Remotes Showing the Difference Between Imperative and Declarative Coding (Yuen, 2021, p. 14).

Today's infrastructure tools, such as Ansible and Terraform, have combined imperative capabilities with declarative languages to support more complex scenarios. This combination allows for the execution of logical operations that cannot be conveyed solely through declarative syntax, offering developers and operators the best of both worlds (Been et al., 2022, pp. 11–15).

2.3 Benefits of DevOps and Infrastructure as Code

IaC enables the provisioning and management of entire infrastructures through scripting and definition files. Its adoption facilitates the accurate and easy replication of systems, departing from the laborious manual methods previously employed. Infrastructure elements, including required software versions and hostnames, are defined within these scripts. IaC fosters dynamic and disposable infrastructures, capable of rapid modification, in contrast to static systems. Changes, such as resizing or relocating resources, are handled with minimal disruption in large-scale cloud ecosystems (Morris, 2021, p. 4).

In one illustrative case, a failure to replicate a manual configuration change from one server environment to another resulted in application failures months later. This incident clearly demonstrates the risks posed by manual processes. In contrast to manual deployment processes, IaC automates the deployment process. Changes to infrastructure scripts trigger a CI pipeline that validates, tests, and deploys updates to the environment, if configured to do so (Humble and Farley, 2010, p. 287).

Defining infrastructure as code and moving it into VCS simplifies the understanding of even the most complex setups, closing knowledge gaps and ensuring better awareness of the

system (Morris, 2021, p. 47). NASA exemplifies transparency by using Ansible Tower to manage its AWS environment, achieving enhanced operations, security, and expedited updates and patching times (Dan London, 2014, p. 1).

The encapsulation of infrastructure within code also allows for continuous testing, ensuring quick verification of any changes. This is vital in dynamic systems where small mistakes can have extensive repercussions. The increased efficiency in software development cycles noted by companies undergoing DevOps and IaC transformations, such as Nordstrom and HP's LaserJet Firmware division, is underpinned by confidence in the system. This has resulted in enhanced feature delivery and reduced defects (Brikman, 2022, p. 23).

IaC now allows for the more straightforward replication of complex infrastructures that traditionally required extensive manual steps. General Motors' use of Chef to automate their ETL platform is a prime example of the benefits of automation. This approach results in faster software deployment and increased productivity (Jeanne Gu, 2019, pp. 1–14).

IaC enables dynamic scaling and distributed architectures. Once a module is written, it can be deployed multiple times with minimal effort, ensuring consistency. The automation and trust developed in the system promotes stability, encouraging more frequent updates and reducing the time developers spend on environment setup.

Additionally, IaC provides cost optimization benefits. Environments should only be deployed when necessary, such as during testing phases, rather than running continuously. This approach yields long-term savings that may otherwise be obscured by seamless application deployments and a stable production environment. Larger organizations, such as GM and CERN, have leveraged Chef and Puppet to manage immense data volumes and software deployments with minimal disruption, illustrating the financial and operational benefits of IaC (A. Rahman et al., 2020, p. 752).

2.4 Challenges of DevOps and Infrastructure as Code

The adoption of Infrastructure as Code and DevOps practices represents a significant change in operational approaches. This transition requires teams to overcome significant skill and knowledge gaps. It demands expertise in cutting-edge tools and the development of a new mindset that considers the opportunity costs associated with resource reallocation and the steep learning curve these methodologies introduce (Brikman, 2022, pp. 384–385). In addition, the abundance of tools available for infrastructure provisioning, configuration, deployment, and orchestration can make it difficult to choose the best option, leading to conflicting choices and a lack of consensus on best practices (Bass et al., 2015, p. 45).

Successful adoption of these practices depends heavily on cultural and organizational dynamics. Resistance to change is a common obstacle, with organizations struggling with issues such as poor communication, entrenched cultures, domain silos, and confusion over DevOps principles. Inconsistent automation can lead to configuration drift, which in turn

results in accumulating technical debt and complicates the adoption process (Morris, 2021, p. 19).

In the realm of IaC and DevOps, project timelines often exceed initial estimates. Timeline extensions in the industry are often caused by factors such as its relative newness, inherent complexities, and common task scope underestimation. Furthermore, the development of infrastructure from scratch is a notably time-consuming task that is often undervalued by planners. These efforts are also prone to "yak shaving", where disproportionate amounts of time and resources are spent on seemingly tangential tasks (Brikman, 2022, pp. 257–259).

Development within IaC, while transformative, is not immune to human errors, which can lead to challenges that are difficult to replicate and fix. Experts caution that systems engineered through IaC can become so complex that even minor changes in scripts can have significant security and stability repercussions. As infrastructure needs evolve, the complexity of IaC applications grows, making the management of such systems more onerous. Furthermore, it is important to be able to identify configuration issues, which is a vital skill that is often lacking in IT teams. For example, in an incident where a Puppet configuration mistake led to data loss highlights the importance of such skills (Incidents/2017-01-18 Labs - Wikitech, 2024).

Additionally, it is crucial to be vigilant in identifying "code smells" - patterns in coding that may indicate potential security issues - as the code developed may introduce security vulnerabilities. Examples such as hard-coded secrets and the use of insecure communication protocols indicate deeper security concerns. Proactive pattern detection through thorough code reviews and automated scanning tools is critical in strengthening the security posture of IaC scripts (Rahman, Parnin, Williams, 2019, pp. 164, 165).

To successfully navigate the complexities of IaC and DevOps, organizations must adopt a holistic approach that considers not only technological solutions but also addresses the human and cultural aspects. The adoption of best practices is paramount, and practitioners must commit to continual learning and improvement, keeping abreast of the evolving landscape of tools and methodologies. Security, often minimized in the rush to adopt new technologies, must be at the forefront of this transformation, with a conscientious effort to inspect, identify, and rectify security smells and vulnerabilities as early as possible in the development lifecycle (Humble and Farley, 2010, p. 278).

3. Methodology

A well-defined methodology is crucial for any thesis as it provides a clear roadmap of the research process, ensuring that the study is conducted systematically. By adhering to a framework, the research not only gains credibility but also allows for the potential replication of the study by other researchers (Peffers et al., 2007, p. 49). This chapter presents the structured approach that is employed in this study to tackle the research questions and to achieve the objectives outlined in Sections 0 and 1.3.

3.1 Introduction to Design Science Research

The objective of this thesis is to migrate an existing IT infrastructure to IaC and to open-source the resulting codebase. This endeavor aims to bridge the research gap defined in Section 1.1 by providing source code examples that can help small to medium-sized enterprises overcome their knowledge deficiencies.

This research is subsequently directly related to the research area of Information Systems (IS), which is defined as "the study of complementary networks of hardware and software that people and organizations use to collect, filter, process, create, and distribute data" (Bourgeois et al., 2019, p. 5). In other words, Information Systems (IS) encompasses various disciplines that bridge the gap between the technical implementation of information technology (IT) and the practical challenges within organizational contexts (Peffers et al., 2007, pp. 46–47).

In the field of IS, quantitative and qualitative research methodologies are employed to provide a contextual understanding of information systems within organizations (Creswell and Plano Clark, 2018). While acknowledging the value of these approaches, the practical orientation of this thesis necessitates a methodology that explores a phenomenon and produces tangible results.

Therefore, the methodology employed in this work is Design Science Research, which is focused on developing and evaluating innovative solutions to organizational problems in IS. As described by Peffers et al. (2007, p. 46) DSR involves creating and evaluating IT artifacts intended to solve identified issues. In the context of DSR, an artifact is defined as any designed object that embodies a solution to a research problem. This can take various forms, such as constructs, models, methods, instantiations, or even social innovations (Peffers et al., 2007, p. 49). For this thesis, creating the source code as an artifact directly addresses the identified research gap that this thesis aims to tackle, making DSR an ideal choice.

3.2 Selecting an Optimal DSR Methodology

When examining notable methodologies in DSR, the methodologies developed by Wieringa, Hevner, and Peffers are frequently selected.

Wieringa's model places significant emphasis on problem investigation and treatment design. Although this approach is systematic, it is less focused on the practical development of IT artifacts. Despite being more appropriate for theoretical exploration of design problems, Wieringa's model is not optimally suited for the practical solutions sought in this thesis (Wieringa, 2014).
On the other hand, Hevner's framework is renowned for its emphasis on the IT artifact and its contribution to knowledge. However, it should be noted that Hevner's framework is based on three design circles for iterating on the artifact, which overcomplicates the process (Hevner et al., 2004).
However, Peffers' approach is particularly well suited. The six-step process it provides allows for rigorous development, demonstration, and validation of the artifact in a sequential and systematic manner. It was ultimately chosen for this thesis because it follows the general structure of a typical undergraduate thesis and allows for a less restrictive approach while still providing clear guidelines for a successful research project (Peffers et al., 2007).

3.3 Implementing Design Science Research Methodology by Peffers

Peffers' Design Science Research Methodology (DSRM) consists of six distinct steps which provide a clear and structured approach that facilitates the progression from problem identification to effective communication of solutions. A thorough understanding of the six steps depicted in Figure 7 is critical for grasping the workings of DSRM and the methodological approach of this thesis.

3.3.1 Problem Identification Step

In DSRM by Peffers, the first step involves identifying the specific research problem and justifying the value of a solution. This sets the stage for a more profound understanding of the problem, motivating both the researcher and the audience to pursue and accept the proposed solution (Peffers et al., 2007, pp. 52–56).
In this thesis, the problem was identified through a literature review, which was detailed in Section 0. A survey was conducted to support the assertion of a research gap. The survey could not only confirm the need to address the problem outlined, but also would provide empirical evidence of its importance in practice. By capturing the views and experiences of individuals closely associated with the problem domain, the survey results would lend credibility to the existence of the problem and the need for a solution.

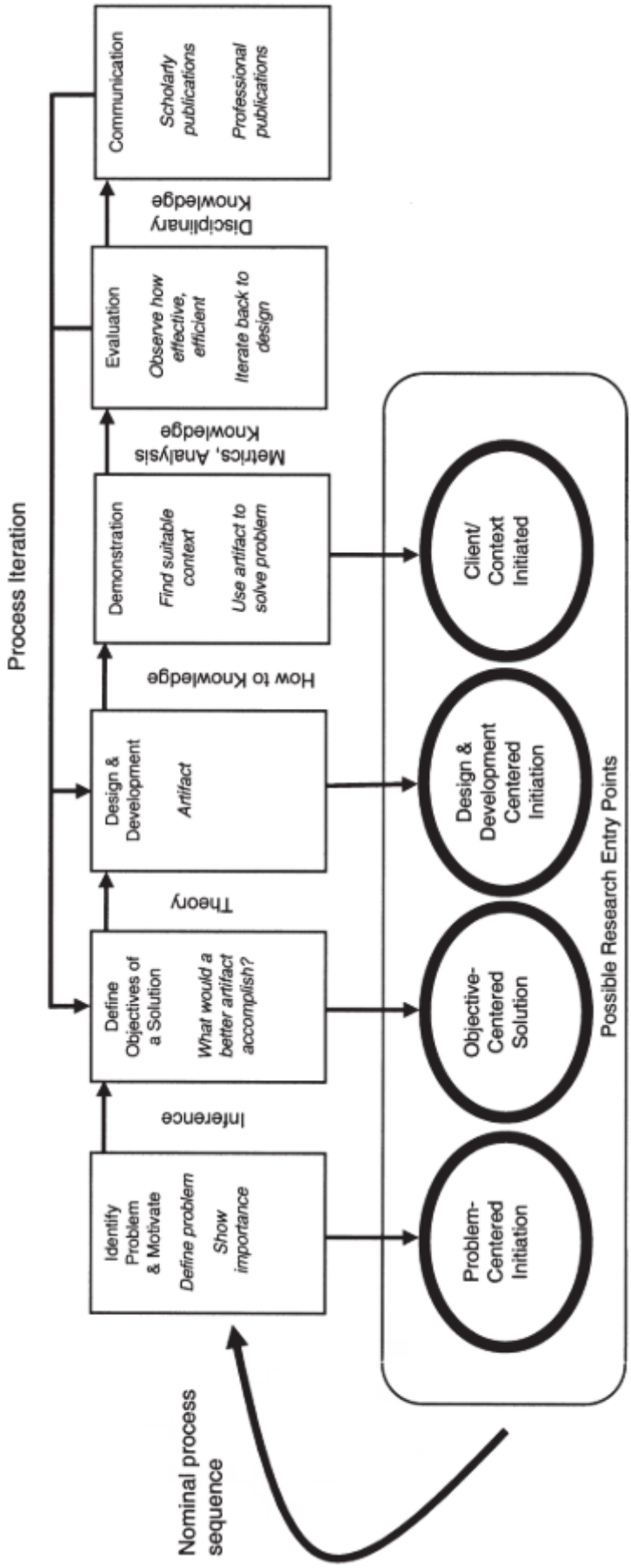


Figure 7: DSRM Process Model by Peffers (Peffers et al., 2007, p. 54).

3.3.2 Objectives of a Solution Step

Once the problem has been identified, the next step involves deriving the objectives for their solution, considering what is both possible and feasible. Objectives can be defined both quantitatively and qualitatively and should be rationally derived from the problem statement (Peffers et al., 2007, pp. 52–56).

To ensure practicality, a descriptive case study method was selected for this step. A case study is considered ideal for this purpose because it allows for an in-depth analysis of a specific IT environment, providing a detailed and practical example for which the artifact can be developed (Creswell, 2009, p. 207; Flyvbjerg, 2006, pp. 223–224).

The selected single case study focuses on a functional IT environment, referred to as a Homelab, which is managed exclusively by the author and is fully understood. The Homelab is a self-contained and controlled setting that is ideal for experimental alterations and rigorous testing. It serves as a microcosm of a typical IT infrastructure and provides a tangible model for illustrating how the source code can be used to operationalize an IT environment (Demir, 2023; What is a Homelab and what does it do? : r/homelab, 2024). When selecting this environment, the author can utilize their knowledge of the system to ensure that the research is informed by direct experience and technical expertise.

However, it was important to validate its standards against broader industry practices to emphasize Homelab's capabilities and relevance as a representative SME model and to address any concerns about subjectivity. Based on structured interviews with three industry experts, a table of typical features of IT environments was created. The functions of the provided Homelab were then compared with that table. This approach allowed for external benchmarks and contextual insights, ensuring that the objectives set for the Homelab would have broader applicability and resonance in the SME space.

To accurately formulate the coherent objectives, it is essential to determine what should be considered most significant to include in the artifact. These priorities were selected based on submissions from the previously mentioned survey, and by assessing what would have the greatest impact on the case study's environment. To further guide the development of the artifact and to ensure that the goals for the artifact are clear, traceable, and achievable, goals are defined using the SMART (Specific, Measurable, Achievable, Relevant, and Time-bound) criteria (Greg Watts and Norman Watts, 2018).

3.3.3 Design and Development Step

With clear goals established, the research moves to the design and development step to create the artifact. This can take the form of a construct, model, method, or instantiation. The transition from objectives to design requires theoretical knowledge that can contribute to the practical realization of the solution (Peffers et al., 2007, pp. 52–56).

Extensive literature research was conducted to acquire the necessary knowledge for designing and developing the Infrastructure as Code environment. Scholarly articles, relevant books, and industry reports were reviewed to gather pertinent information. The key learnings that informed the development of the IaC environment are summarized in Chapter 2 and practically applied in Chapter 6.

3.3.4 Demonstration Step

The fourth step is a demonstration, which serves as a proof of concept and presents the artifact as a viable solution to the problem. This section outlines the use of the artifact in different contexts, including experiments, case studies, or proofs (Peffers et al., 2007, pp. 52–56).

Due to the limitations of the thesis scope, it is not possible to demonstrate all artifact functionalities. However, the core functionalities will be described with a focus on selected features in the context of the case study. The descriptions aim to offer practical and empirical evidence of how the artifact can be utilized by SMEs, providing a clear understanding of the practical applications of IaC and ultimately explain how the research results were obtained.

3.3.5 Evaluation Step

During the evaluation step, the artifact's performance will be observed and measured against the solution objectives that were defined earlier. This will involve comparing the observed results from the artifact's use in the demonstration with the objectives that were defined earlier. The evaluation may include quantitative performance measures, client feedback, or any other suitable empirical evidence (Peffers et al., 2007, pp. 52–56).

The evaluation in this thesis will involve a comparative analysis between the specified objectives from Chapter 5 and the results developed in Chapter 6. The artifact will be assessed by utilizing the established monitoring in the case study environment and manually verifying the functionality of each service post-migration. This allows for the evaluation of the artifact and to determine whether it meets the predefined objectives.

3.3.6 Communication Step

The final step is communication, which involves sharing the knowledge gained from the research process. This includes communicating the problem, the artifact, its utility, and the effectiveness of the design (Peffers et al., 2007).

For this study, the complete source code, including this thesis, will be made publicly available on GitHub. To increase visibility this thesis will be promoted in relevant Reddit communities and in a personal blog post. Finally, the complete thesis will be presented in a public thesis defense. Despite limited resources and formal publishing barriers, this strategy has been crafted to efficiently share the research with a broad audience.

4. Assessing the Need for Extensive Open-Source IaC Artifacts

The first chapter of this thesis defined the problem scope identified through a literature review, revealed a significant gap. Although there were many generic examples, there was a noticeable lack of comprehensive source code examples that demonstrate a complete deployment of an IT environment using IaC. To analyze the limited practical applications of literature and identify the problem in accordance with the problem identification step for DSRM, a structured survey was conducted to research the need for a practical artifact.
The survey's complete rationale for its structure and questions will not be highlighted in this chapter, as the main objective is to develop an artifact. However, the survey also inquired about the most popular IaC tools and critical features for an IaC artifact. This information will later be used to take more informed decisions. Therefore, this chapter will also provide sufficient background information to validate the survey results.

The survey was created using LimeSurvey and was deployed within the case study environment. It was promoted on /r/homelab, a community with over 650,000 IT enthusiasts, including system administrators and developers. The dissemination strategy was successful, resulting in over 100 expressions of interest and 81 survey responses, although 28 were incomplete. The complete survey results have been attached to this thesis as a LimeSurvey PDF export, while a public summary of the findings and the Reddit post can be found at these URLs:

- https://www.reddit.com/r/homelab/comments/1bonfz4/unveiling_my_mini_but_mighty_homelab/
- https://survey.sauna.re/index.php/statistics_user/661976

The preliminary section of the survey aimed to validate the participant pool by asking about their professional backgrounds, tenure in the IT sector, and proficiency with IaC tools. The survey results indicate a diverse range of job roles within the IT industry. Cyber Security Specialists, IT Managers, and Full-Stack Developers comprise approximately 27% of the respondents.
Industry experience ranged from individuals with no IT background (7.41%) to those with over a decade of experience in the field (32.10%), who were the largest respondent group. Upon closer examination of the data, it was found that most respondents (60.5%) had beginner to intermediate skill levels with IaC tools.
Interpreting the respondents' background, the survey successfully captured a representative sample of the IT industry, with a broad spectrum of roles and experience levels, which confirms the relevance of the participant pool to the field of Infrastructure as Code.

Further examination of the responses revealed that 32.09% of participants rated the availability of open-source IaC deployments as “Very poor” or “Poor”. Additionally, 44.45% of the respondents rated the comprehensiveness of existing open-source IaC code examples as “Not at all comprehensive” or “Slightly comprehensive”. Only a minority considered the available resources as “Very comprehensive” (7.41%), with no participants rating them as “Fully comprehensive”.

In conclusion, the data suggests that open-source Infrastructure as Code deployments are currently insufficient for the IT industry's needs. Participants reached a consensus that these tools lack the necessary depth to be considered comprehensive or fully satisfactory. Therefore, there is a clear need for enhanced, detailed, and widely available IaC examples, which supports the motivation for this research and the proposed creation of a practical IaC artifact as stated in the problem statement.

.

5. Formulating the Objectives of the Artifact in the Case Study

As explained in Chapter 3, the objectives of the artifacts will be derived from a case study. Following this rationale, a Homelab environment was chosen as the appropriate setting for this research. This chapter validates the relevance of the Homelab environment to SMEs. As it is a descriptive case study it then provides a detailed description of the Homelab environment. Once this basic understanding is established, the appropriate automation tools are selected and the objectives for the artifact are formulated based on the case study requirements and survey results.

Although it is beyond the scope of this thesis to detail the full functionality of the environment, the insights provided in this chapter are instrumental in clarifying both the nature of the development undertaken and the rationale behind it.

5.1 Suitability of the Homelab Environment

The investigation of the selected Homelab environment is a crucial preparatory step to ensure the relevance and applicability of the case study. It is necessary to prove that the Homelab environment is a valid representation of an SME's typical operational setting. To establish this validity, a systematic approach was adopted, which involved conducting interviews with industry professionals.

The study's interviewees were [redacted], a systems engineer with seven years of experience at [redacted] and [redacted] and [redacted], both senior systems engineers at [redacted] with 25 and 24 years of experience, respectively. Each expert was interviewed twice individually via Microsoft Teams for approximately one hour per session. As the interviews serve as a supplementary means to support the main focus of the research, which is the development of functional IaC source code, they were not transcribed.

In the first round, the interviewees were asked to brainstorm and enumerate all functions that an IT department routinely administers. The participants were prompted to consider the various tasks that a systems engineer may undertake. As the functions were listed, they were promptly recorded, and follow-up questions were asked to obtain specific examples of products associated with each function.

Subsequently, the collected information was synthesized, and similar or duplicate functions were consolidated based on their commonalities. The objective was to create a comprehensive table of functions without excessive detail, resulting in 35 final distinct functions.

Consequently, the table containing the functions and product examples was shared with each interviewee at least 72 hours prior to the second interview, allowing sufficient time for review and potential amendments. However, none of the interviewees suggested any corrections.

During the second interview, participants were asked to rate the frequency of each function utilized in SMEs using a standardized scale ranging from 1 (minimal), 2 (moderate), 3 (common), 4 (high) to 5 (extensive). The responses were compiled, and an average rating was calculated for each function. The averages were rounded to the nearest whole number following the standard convention of rounding up when the decimal was .5 or higher and rounding down when it was less than .5.
Those calculated frequency ratings were then added to the table as a new column titled “Use in SME”.
The next step was to identify the IT functions present in the case study environment and document the service that fulfills each function in a newly created “Deployed in Case Study” column. If the function is not being fulfilled, it was denoted with a slash.
The finalized table, containing the functions, product examples, frequency ratings, and their corresponding applications within the case study environment, can be found in the Appendix Table 3.
This systematic alignment of functions with their corresponding applications allowed for a measurable evaluation of the Homelab environment's capabilities. Analysis revealed that out of the 35 identified functions, 23 were present in the Homelab environment. Upon closer evaluation of functions with a frequency rating of "common" (value 3) or higher and excluding those marked as “minimal” (value 1) and “moderate” (value 2), it was found that 21 out of 29 such functions had been implemented, amounting to approximately 72%. To clearly illustrate this, a condensed table applying this filter has been included as Table 1 to this chapter.
The case study environment does not include specialized applications such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Human Resources Management (HRM), and Endpoint Management. These applications typically involve the use of complex platforms from providers like Microsoft Dynamics, SAP, and Salesforce, which are currently impossible to automate within the scope of this project. As a result, their absence from the Homelab environment does not affect its usefulness as a model for an SME's IT infrastructure.

Based on the data collected from interviews and subsequent analysis, the resulting Table 1 confirms that the selected case study environment is well-equipped to replicate the typical operational IT landscape of an SME.

Table 1: Consolidated IT Functions in SMEs and Their Presence in the Homelab Case Study.

Function	Use in SME	Deployed
Groupware and Email	Extensive	Exchange Online, Microsoft 365
Team Collaboration Messaging	High	Microsoft 365
Accounting and Financial Management	Extensive	IHateMoney
CMS	High	Ghost
CRM	Common	/
ERP	Common	/
HRM	Common	/
Endpoint	Extensive	Multiple Devices
Server Computer	Common	Custom Build Server
Switching	Extensive	Cisco
Telephone	Extensive	FRITZ!Box
UPS	Common	BlueWalker
Wi-Fi Access Point	High	Ubiquiti
Endpoint and Server Management	High	/
Hypervisor	Common	Proxmox Virtual Environment
IT Ticket Management	Common	/
Print Server	Common	/
System Monitoring and Logging	Extensive	Prometheus, Uptime Kuma
Data Backup and Recovery	Extensive	Proxmox Backup Server, Rsync
DMS	High	Paperless-ngx, Calibre
Network Storage	Extensive	OpenMediaVault, Filebrowser
DHCP and DNS	Extensive	AdGuard Home
SIP Gateway	High	/
VPN	High	WireGuard
Web Server and Reverse Proxy	Extensive	Traefik, NodeJS, Apache
Antivirus Software	Common	/
Domain Management and IAM	Extensive	Microsoft Entra ID, Authelia
NGFW	Extensive	OPNsense
Password Management	High	Vaultwarden

5.2 Hardware and Network Setup

A hybrid cloud is a computing environment that combines on-premise and public cloud services (Rich Mogull et al., 2017, pp. 11–12). This case study uses such an environment, which utilizes the advantages of both on-premises and cloud resources to optimize operational efficiency. This section will provide additional details about that environment.

The on-premises infrastructure, shown below the dotted line in Figure 8, is anchored by a Proxmox Virtual Environment server. This hypervisor hosts a combination of five virtual machines (VMs) and two Linux Containers (LXC). The virtual machines include OPNsense, an open-source next-generation firewall (NGFW) that serves as the cornerstone for network security. OPNsense is responsible for DHCP management, VPN server, VLAN management, routing, and network security measures, including intrusion detection, IP blocking, and firewall rules.

Two additional virtual machines are used to meet storage and service hosting requirements. The first VM runs Openmediavault and provides network-attached storage (NAS) using solid-state (SSD) and hard disk drives (HDD). The second VM runs Ubuntu Server and is located within a demilitarized zone (DMZ) network to ensure a secure connection from public sources. In both VMs, Docker and Docker Compose are used for service deployment. The infrastructure's backup solution is managed by Proxmox Backup Server, which uses virtual machine snapshots to perform both incremental and full backups. The final virtual machine is Home Assistant, an open-source project on GitHub that manages approximately 40 Internet of Things (IoT) devices.

Network communication is handled through a Cisco 8-Port Switch, which interfaces with a FRITZ!Box Modem to provide OPNsense with internet access. In addition, a Ubiquity Access Point (AP) provides Wi-Fi coverage for devices in the LAN and IoT networks. VLAN tags are used to manage and segregate network traffic, and a pair of authoritative DNS servers operate within LXC containers to manage network naming. An uninterruptible power supply (UPS) is a critical component that ensures the continuous operation of infrastructure components during power outages. It is the final hardware piece on-premises.

Moving on to the cloud-based infrastructure components portrayed above the dotted line in Figure 8. The case study includes both Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) elements. Google Cloud Storage is used as an off-site backup solution, while Azure Web App hosts a web portal for a non-profit organization. IaaS components consist of a virtual private server (VPS) from 1blu and two VMs hosted on Azure. All three components deploy services using Docker and Docker Compose.

Refer to Table 4 in the Appendix for a comprehensive list of all deployed services. However, Table 2, which is a condensed version, will be used throughout this thesis.

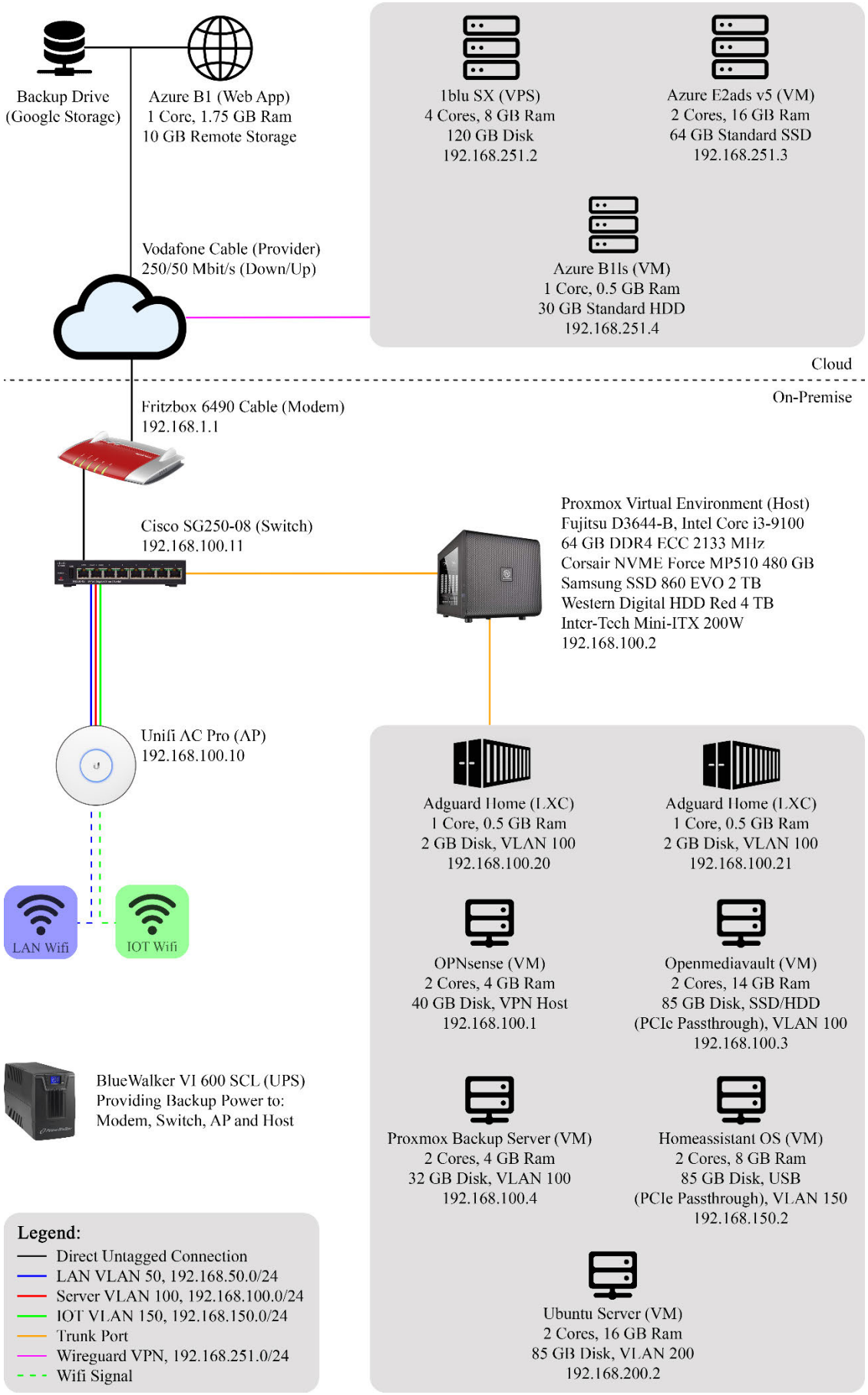


Figure 8: Overview of the Hardware and Network Setup (Own Illustration, 2024).

Table 2: Condensed Overview of Deployed Services, Including Automation Level, and Status as of the Submission Date.

Application	Automation Level	Status
Adguard Exporter	Fully	Done
Adguard Home*	Fully	Pending
AdguardSync	Fully	Done
Alertmanager	Fully	Done
Apache	Partially	Pending
Authelia	Fully	Pending
Autoscan	Fully	Pending
Azure Entra ID*	Manual	Excluded
Backup Backend*	Partially	Pending
Backup Solution*	Partially	Pending
Bazarr Exporter	Fully	Done
BetterGPT	Partially	Pending
cAdvisor	Fully	Done
Calibre	Fully	Pending
Calibre-Web	Fully	Pending
Certdumper	Fully	Done
Changedetection	Partially	Done
Cloudplow	Fully	Pending
Code-Server	Partially	Done
Czkawka	Fully	Done
DHCP*	Fully	Pending
Docker-Socket-Proxy	Fully	Done
Dockge	Partially	Done
Exchange Online*	Manual	Excluded
Filebrowser	Partially	Pending
Ghost	Partially	Pending
Grafana	Partially	Done
Headscale	Partially	Pending
Homer	Fully	Pending
IHateMoney	Partially	Pending
Librespeed	Fully	Done
Librespeed Exporter	Fully	Done
Linkstack	Fully	Pending
Microsoft 365*	Manual	Excluded
NAS Server*	Partially	Pending
Node Exporter	Fully	Done
NodeJS*	Partially	Pending
NTP*	Fully	Pending
Overseerr	Partially	Pending
Paperless-ngx	Partially	Done

Application	Automation Level	Status
Photoprism	Partially	Pending
phpMyAdmin	Fully	Pending
Plausible	Partially	Pending
Plausible Exporter	Fully	Done
Plex	Partially	Pending
Plex Exporter	Fully	Done
Portainer	Partially	Done
Portainer Agent	Fully	Pending
Prometheus	Fully	Pending
Prowlarr	Partially	Pending
Prowlarr Exporter	Fully	Done
Proxmox Exporter	Fully	Done
pyLoad	Partially	Done
qBittorrent Exporter	Fully	Done
Radarr	Partially	Pending
Radarr Exporter	Fully	Done
Rclone	Fully	Pending
Recyclarr	Fully	Pending
Routing*	Fully	Done
Sabnzbd	Partially	Pending
Sabnzbd Exporter	Fully	Done
Searxng	Fully	Pending
Smart Home Server*	Partially	Pending
Socks5-Proxy	Fully	Done
Sonarr	Partially	Pending
Sonarr Exporter	Fully	Done
Tandoor Recipes	Fully	Done
Tautulli	Fully	Pending
Traefik	Fully	Done
Unifi Controller	Partially	Done
Unifi Exporter	Fully	Done
Uptimekuma	Fully	Pending
Valetudo	Fully	Done
Vaultwarden	Fully	Pending
Watchtower	Fully	Done
WireGuard VPN*	Fully	Pending
Wizarr	Partially	Pending
xBackBone	Fully	Pending
Xteve	Fully	Pending
Your Spotify	Fully	Pending

5.3 Choosing the Optimal Provisioning and Configuration Management Tools

Now that the hardware, technologies, and services have been described, it is crucial to assess their potential for automation to formulate concrete objectives for the artifact. Thus, it is crucial to first choose the right tools for provisioning and configuring the infrastructure and comprehend their capabilities before assessing the objectives of the artifact.

A decision-making process is necessary to select the most suitable tools for provisioning and configuration management. The infrastructure includes Proxmox Virtual Machines, Google Cloud Storage and Azure Web Apps as PaaS offering, and Virtual Machines hosted on Azure. Considering the components at hand, it may be worthwhile to explore native options such as Azure Resource Manager for Azure and Google Cloud Deployment Manager for Google Cloud services (Wurster et al., 2020, pp. 66–67). It is worth noting that Proxmox does not currently provide a proprietary tool for these purposes. To address the requirement for deployment across various clouds and the local hypervisor, the options are narrowed down to either adopting multiple specialized tools or selecting a single, versatile tool capable of managing heterogeneous hybrid cloud environments. After careful consideration, it has been decided that due to the increased simplicity of learning just one tool, choosing the latter option would be the best course of action. Therefore, Terraform, a tool that supports provisioning the hardware components of the hybrid cloud environment at hand, has emerged as the preferred tool (Brikman, 2022, p. 39).

When considering a general-purpose configuration management tool, there are several options available, including Ansible, Chef, Puppet, and Salt (Brikman, 2022, pp. 39–47). Each of these tools proves its own strengths that could effectively meet the needs of this thesis. However, it is important to consider factors beyond just functional capabilities when selecting. Popularity, ease of use, and underlying technology dependencies should also be considered. For this case study, Ansible was chosen as the configuration management tool because it is the most popular configuration tool according to the survey conducted, with 49.38% of respondents using it. Additionally, Ansible is unique in its ability to operate on target machines without an agent, and it is implemented using Python, a widely used programming language (Stack Overflow, 2024).

5.4 Automation Potential for the Case Study’s Services

The following step involves evaluating the automation potential of each service engaged in the case study when using Ansible and Terraform. This evaluation is crucial for setting achievable objectives and providing a genuine appraisal of the extent to which the environment can be automated.

The "Automation Level" column in both Table 2 (condensed version) and Table 4 (full version) reflects the level of automation for each service. This level is derived from the capabilities of the tools under consideration. Due to the limited scope of this paper, a more detailed examination of the automation levels is not provided.

The “Automation Level” is categorized into three tiers:

1. Fully Automated:
   This tier includes service provisioning and complete service configuration.
   Provisioning involves setting up the infrastructure by creating virtual machines and installing applications to their default state.
   Following this, service configuration begins. Administrators must customize the default installations to ensure that the services function as intended within the unique operational context of the organization. This implies that all components of an application's setup, including its distinct settings and preferences, can be duplicated to match the state it held before migration using only Terraform and Ansible.
   This level of automation is especially attainable for applications that store their configuration in editable file formats such as JSON or YAML, or that manage configuration through environment variables. These formats allow Ansible to modify settings without requiring direct human intervention.
   The case study shows that 49 out of 80 services achieved full automation.
2. Partially Automated
   Services are categorized as partially automated when only the provisioning can be automated, but configuration cannot be completed through the selected tools.
   In this case study, 28 out of the 80 unique services fell under the partial automation category.
3. Manual
   This level is assigned to services that cannot be provisioned or configured using Terraform or Ansible.
   In this case study, mainly Microsoft's SaaS offerings fall under the manual category. Services like Microsoft 365, Azure Entra ID (formerly Azure AD), and Exchange Online are fully managed by Microsoft and cannot be automated using the selected tools. Although PowerShell can be used to administer these services, Terraform and Ansible were chosen for their extensive adoption and ease of use. Therefore, PowerShell-based processes were not included in this thesis. Because these services are incompatible with the tools selected for this thesis, they are not shown in Figure 8. Instead, to ensure comprehensiveness they are documented in both services overview tables (Table 2, Table 4).

5.5 Determining the Artifact's Features

To effectively manage an IaC project, it is essential to understand two key aspects. First, one must understand the theoretical potential for automation, as outlined in the previous section. Second, it is imperative to consider the time frame available and the practical limitations of implementing automation within that time frame. Proper prioritization is, therefore, critical. The survey conducted provides guidance for this step, which will be discussed in this section.

This thesis was allocated three months. The first half was dedicated to migrating to Infrastructure as Code, and the latter half devoted to composing this thesis. Despite the need to learn and apply new tools, achieving full completion was unfeasible due to the complexity of the environment and the time constraint.

The primary objective was to transfer as many services as possible within the allotted time, giving priority to areas with the highest impact and significance based on the setting and responses to the survey question: "Select the IT environment capabilities you consider most critical to be included in IaC code deployment.". This approach of output maximization would also provide a realistic assessment of what could be achieved within the limited time, offering valuable insights into the field.

The main philosophy of the migration is to implement the practice of "Data Normalization" known from database management. This practice reduces redundant information, enhancing efficiency and reducing the risk of human error. Table 2 shows that of the 80 deployed services, 13 are marked with an asterisk, indicating that they are not deployed using Docker. As the majority uses Docker, the initial focus was on automating Docker services. Creating a template to deploy and orchestrate all future Docker containers ensures uniformity across all instances while maintaining necessary flexibility. According to the conducted survey 39% voted this capability as the 4th most critical.

Another important aspect was consolidating URL configurations, which often lead to redundancy. Each service requires a unique URL that must be specified in the service configuration, reverse proxy, and either in one or two monitoring systems. The survey, which showed over 50% agreement with automated monitoring, underscored the need for a centralized URL management system, not only because it was voted the most critical requirement, but also because of the prevalence of outdated entries and the omission of new entries in the case study monitoring systems. This new system would serve as a singular source of truth, automatically applying this information across service configurations, reverse proxies, and monitoring systems. This consolidation would streamline processes and contribute to the objective evaluation of whether IaC-migrated services were functioning as intended, which adds groundwork for the evaluation step from the DSRM by Peffers which will be discussed in Chapter 8.

The survey showed that the second most requested feature was automatic patch management. However, the case study has this feature already addressed by enabling unattended upgrades through Apt and Watchtower. In addition, there are publicly available Ansible examples for this feature. As a result, attention shifted to the third most requested feature: automated server provisioning. By using Terraform, this feature would allow for automated scaling and the elimination of snowflake systems, ensuring consistent configurations across operating systems.

5.6 SMART Objectives for the Artifact's Development

The objective is to create an artifact that implements IaC in a way that is relevant and practical for SMEs. Due to time constraints, it is not feasible to fully migrate an entire case study environment to IaC. To ensure that the objectives are relevant to SMEs and achievable within the given timeframe, the SMART framework is utilized. This framework outlines that objectives should be Specific, Measurable, Achievable, Relevant, and Time-bound to ensure they are clear and attainable within a certain timeframe (Bovend'Eerdt et al., 2009).
Based on the detailed requirements outlined in this chapter regarding the artifact's features within the context of the case study, the following objectives have been selected:

Objective 1:

- Specific: Create a template that can automatically deploy all existing Docker services using Ansible.
- Measurable: Track the number of services successfully migrated using the template and confirm that they are operational through monitoring and manual checks.
- Achievable: Due to the large number of Docker services, migrating all of them to IaC may be challenging. However, it is not necessary to deploy all services to verify their functionality.
- Relevant: Over 80% of all services are deployed using Docker (67 out of 80), Docker Compose is the most used IaC tool according to the survey, and container orchestration is the fourth most requested feature at 39%.
- Time-bound: The template must be created within three weeks, and migration should be completed for as many services as possible within half a week.

Objective 2:

- Specific: Implement a centralized URL management system that defines URLs only once and automatically populates the URLs across service configurations and monitoring systems, to create an extensive and scalable monitoring system.
- Measurable: The repository's code can be searched for duplicated or hard-coded URL entries.

- Achievable: With Ansible, variables can be defined and, using templating, inserted into configuration files.
- Relevant: It reduces redundancy, lowers the potential for errors, and enables automated monitoring, the most requested feature in the survey.
- Time-bound: The centralized URL management system is expected to be operational within two and a half weeks.

Objective 3:

- Specific: Import all current VMs and LXCs into Terraform to allow the infrastructure management through Terraform Code.
- Measurable: There are no VMs or LXCs that are not managed by Terraform.
- Achievable: Automatic provisioning would necessitate recreating each VM and LXC from scratch. Terraform would be responsible for provisioning templated servers, while Ansible would configure them to completion. Given that time is the greatest constraint, the immediate objective is limited to managing infrastructure through Terraform, rather than achieving full automation of server provisioning.
- Relevant: In the survey, automatic server provisioning was the third most requested feature. It simplifies scaling and resolves issues with snowflake systems.
- Time-bound: The objective of importing the complete infrastructure must be achieved within one week.

6. Developing the IaC Source Code

This thesis aims to address the gap in openly available Infrastructure as Code examples. In the previous section, a systematic approach was presented to extract the fundamental requirements for the artifact, resulting in three specific objectives. The following section focuses on both the Design and Development and the Demonstration step of the DSRM by Peffers. By explaining the reasoning behind the artifact's design and development the code and its effects will be demonstrated.

The artifact's development is intentionally directed towards fulfilling the stated objectives and addressing the research question by providing valuable source code. Leveraging the insights from Chapter 2 the IaC project will be developed. The knowledge gained from this chapter will benefit others who are pursuing similar endeavors.

This section will first illustrate how to meet the prerequisites for successful IaC development, followed by a detailed execution of each of the three objectives.

6.1 Getting Started with Infrastructure as Code

This subchapter covers the necessary steps for creating a functional development environment to maintain IaC source code. It begins with a brief system preparation, installation of required software, and then proceeds to describe the setup of the IaC repository. These steps are essential for anyone who wishes to begin an IaC project or reuse the case study's environment.

6.1.1 Establishing the Development Environment

A crucial limitation of Ansible is that it cannot be natively executed on Windows environments. This posed an initial challenge for the development process in this case study. To address this issue, the implementation followed Ansible's guidance by opting for a Windows Subsystem for Linux (WSL) installation (Installing Ansible — Ansible Community Documentation, 2024). Another alternative would be to create a Linux-based machine and use it as an Ansible jump host for development.

After installing Ansible according to official guidelines, Ansible Lint was configured to identify errors and enforce best practices in Ansible files (Installing - Ansible Lint Documentation, 2024).

In contrast to Ansible, Terraform natively supports Windows. The operational environment for Terraform was established by following the “Get Started – Azure” guide from Terraform, which ensured proper integration within the Azure ecosystem (Azure | Terraform | HashiCorp Developer, 2024).

Continuing with Version control, a crucial DevOps practice that ensures the integrity of the development process, Git was installed on the Windows system, and the repository was initiated using “`git init`” through the Windows terminal. Further configuration was

required to customize the setup with unique identifiers containing the collaborators' information for each commit. This was achieved by setting the global user ID using the command "`git config --global user.name 'Fabio Sauna'`" and "`git config --global user.email github@sauna.re'`". Once changes to the code are committed locally, they are then pushed to a remote repository created and hosted on GitHub. This synchronization not only ensures an off-site backup to safeguard the codebase, but also allows for public access to the source code, which forms the base of the communication step. The choice of the Integrated Development Environment (IDE) is subjective and influenced by individual preference. However, Visual Studio Code (VScode) was chosen for this study due to its robust features and lightweight design. To ensure a seamless development experience, several extensions were installed, including the "Remote Development" extension pack from Microsoft, which allowed VScode to interface directly with WSL, the "Git Extensions Pack" for improving the source control management. In addition, various syntax highlighting extensions for Terraform, Ansible, and Jinja2 were added to enhance ease of coding.

6.1.2 Repository Setup for Ansible and Terraform

After installing the necessary software to work effectively with Ansible and Terraform, a structured and configured repository is required. This section explains the main aspects of the case studies repository and its contents, which are shown in Figure 9.

The primary configuration file for Ansible is the ansible.cfg file, which defines the default settings and behaviors of Ansible operations. Parameters such as the default inventory and role path were also considered. The `ANSIBLE_CONFIG` environment variable was set to `/root/vscode/workspace/homelab/ansible.cfg` to ensure that Ansible consistently uses the project-specific settings rather than global defaults.
The inventory, which lists all the nodes from the case study's environment, is a file named hosts.ini. The nodes are categorized into groups and their respective IP addresses, or domain names. This enables Ansible to accurately target and manage each machine. An excerpt of the inventory can be seen in Figure 10.
For authentication purposes, a user "ansible" was established on each host with public key authentication enabled. The corresponding SSH private key, which facilitates secure logins to the managed hosts, was securely stored, and its path was referenced in the inventory file.
The playbook directory contains Ansible playbooks, which are YAML files that define the desired state for each group or host in the inventory. These playbooks consist of multiple roles and tasks, each designed to configure specific aspects of the system, such as applying specific Docker service roles and synchronizing the monitoring configuration of Uptime Kuma.

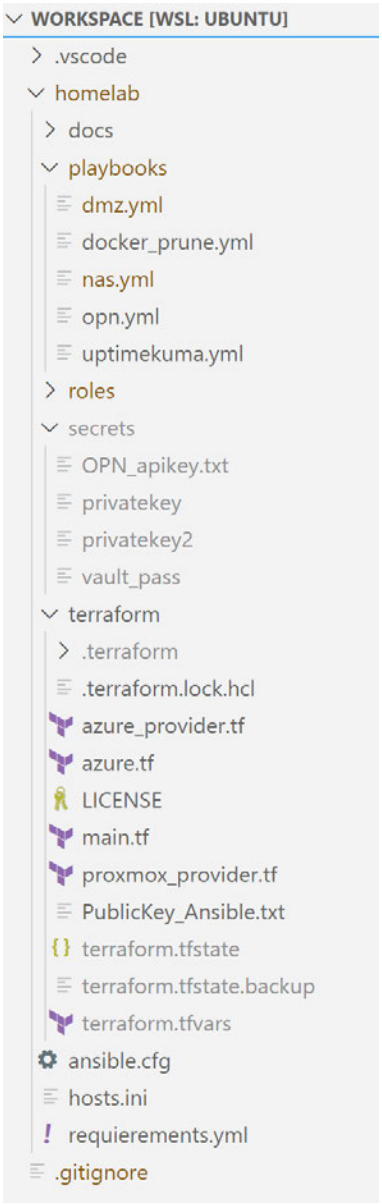


Figure 9: Overview of the IaC Repository (Own Illustration).

homelab > hosts.ini

You, 4 weeks ago | 2 authors (Fabio Sauna and others)

```
1   [all:vars]
2   ansible_user="ansible"
3   ansible_ssh_private_key_file= /root/vscode/workspace/homelab/secrets/privatekey
4
5   [azure]
6   azure ansible_host=172.187.202.93
7
8   [blue]
9   blue ansible_host=195.90.219.248
10
11  [dmz]
12  dmz ansible_host=192.168.200.2
13
14  [nas]
15  nas ansible_host=192.168.100.3
```

Figure 10: Excerpt of the Ansible hosts.ini File (Own Illustration).

When transitioning to Terraform configuration, the requirements are notably less complex than those of Ansible. At the time of writing, the case study uses a pre-release version of the Terraform Provider for Proxmox (GitHub, 2024a). This plugin connects Terraform with the Proxmox hypervisor and currently requires an unofficial manual installation. The decision to use the pre-release version was made to future-proof the Terraform configuration and integrate advanced but required settings. Once the developer officially releases the new version, a description of the installation process of the pre-release version will become obsolete; therefore, only this disclaimer is provided.
Sensitive data, such as SSH private keys, API keys, usernames, and passwords, are stored in specific files and directories. To maintain the security of confidential information, its paths are listed in the .gitignore file, located in the project's root directory. This measure prevents accidental leaks of sensitive data to the remote GitHub repository. Please refer to Figure 11 for the complete contents of the .gitignore file.

Figure 11: Complete Contents of the .gitignore File (Own Illustration).

6.2 Objective 1: Templating the Docker Compose Deployment

This section discusses the process of templating the Docker Compose deployment in the case study. It first examines the existing challenges associated with the Docker Compose configurations, identifies a suitable resolution strategy, and elaborates on that strategy.

6.2.1 Analyzing Existing Docker Compose Challenges

Using Grafana's configuration, shown in Figure 13, as an example, the current challenges with Docker Compose deployments will be analyzed and explained in this subsection.

Every Docker Compose file begins by declaring the version of Docker Compose it adheres to. Within these files, the services block details of the individual containers that make up the service. These containers are each characterized by a collection of attributes within that service's section.
These attributes include the service name, which provides a reference to the containers within the Docker Compose file, as well as the “hostname” and “container_name”, which replace the randomly generated names automatically assigned by Docker. A custom

“hostname” is particularly beneficial, allowing other containers to connect via Docker's internal DNS using an easy-to-remember hostname.
Additionally, containers within Docker Compose can be configured with restart policies that determine their behavior upon exit. In the case study, the “unless-stopped” policy is universally applied to all services. Similarly, the “security_opt” setting, which is usually configured through the complete case study environment to “no-new-privileges”, can be used to prevent processes within the container from obtaining additional privileges, thereby enhancing security (Docker Documentation, 2024a).
Extending this trend of redundancy, the case study uses Traefik as a reverse proxy. Within the Docker environment, Traefik acts as an intermediary for client requests, directing them to the appropriate backend container based on predefined rules, and preventing direct exposure of the container to the network, which is depicted in Figure 12.
In the case study, Traefik’s configuration requires two routers for the two distinct entrypoints. In Traefik terminology, an entrypoint is a definition of network ports that listen for incoming traffic. For example, when the URL “http://grafana.oiba.de” is accessed, it hits the corresponding HTTP entrypoint on port 80. Then, leveraging the “https-redirect” middleware, it is redirected to the secure HTTPS entrypoint on port 443. Middleware in this context refers to a set of commands or services that process incoming requests before reaching the backend service. The section on HTTPS configuration specifies the SSL certificate resolver, which in this case is “acme”, to encrypt the web traffic. It also includes middleware chains for security enhancements, such as IP whitelisting, Single Sign-On with Authelia, HTTP Strict Transport Security (HSTS), and other security headers. Furthermore, the configuration specifies the Traefik service and designates port 3000 as the target port for forwarding traffic to the Grafana container.
To integrate Traefik with Docker services, such as Grafana, it interfaces with the Docker socket to retrieve labels from each container. Traefik functions optimally only with Docker services that have accurate labels. Although the labels offer advanced configuration that adheres to a specific format, they will not be covered here to stay within the scope of this thesis. It is worth pointing out that most services share a very similar configuration (Traefik.io, 2024b).

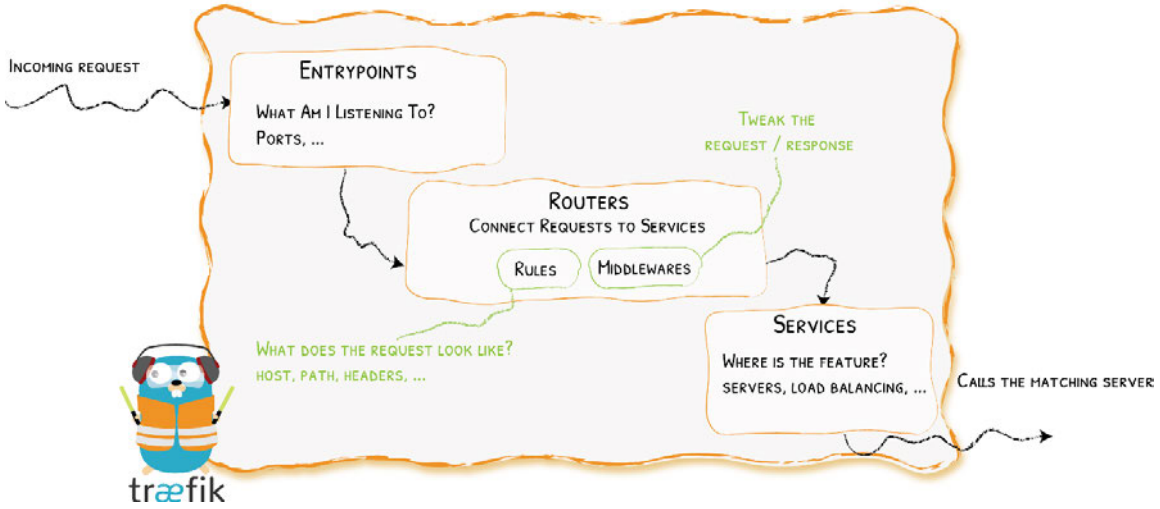


Figure 12: Schematic Overview of Traefik Functionality (Traefik.io, 2024a).

In the Grafana example shown in Figure 13, the last label is used for Watchtower, an application that automates updates for Docker containers. By setting “watchtower=enable” on containers, Watchtower periodically checks for and applies updates to the Docker images (Introduction - Watchtower, 2023).

The final step in understanding Grafana's Docker Compose is to establish networking. To ensure that only containers within the same network can communicate with each other, each container and the overall Docker stack require network definitions (Docker Documentation, 2024b). For Traefik to function correctly, it must be located on the same network as the containers to which it forwards traffic.

To enhance security and limit potential vulnerabilities, dedicated networks are used in this case study. These networks follow a 172.30.xxx.0/24 subnetting scheme, with incremental adjustments to the third octet for each service. Before the IaC migration, attaching Traefik to the network of a newly created service was often overlooked, resulting in gateway timeout errors.

After analyzing the Docker Compose configurations, some aspects of the setup result in inefficiencies. For instance, if the Docker Compose version changes or default values, such as the restart policy setting, need adjustment, the same changes will be necessary across all Docker services. The redundancy of Traefik label configurations is particularly pronounced in this issue. The routing and middleware directives for each service are replicated with little variation, indicating the necessity for a more dynamic approach to managing these settings.

Upon comparing Figure 13 and Figure 14, it becomes evident that the Docker Compose files for Grafana and Portainer have a significant amount of overlap. Most of the configuration is identical, with only a few service-specific settings differing.

This pattern of redundancy is not only present with Portainer, but with most of the 67 Docker services in this study. This fact emphasizes the need for a more efficient approach through the correct implementation of IaC.

```yaml
version: '3.8'
services:
  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    hostname: grafana
    restart: unless-stopped
    security_opt:
      - no-new-privileges
    user: 2000:2000
    networks:
      - monitoring
    volumes:
      - ./data/lib:/var/lib/grafana
      - ./data/log:/var/log/grafana
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=monitoring"
      - "traefik.http.routers.grafana.entrypoints=http"
      - "traefik.http.routers.grafana.rule=Host(`grafana.oiba.de`)"
      - "traefik.http.routers.grafana.middlewares=https-redirect@file"
      - "traefik.http.routers.grafana.service=grafana-https"
      - "traefik.http.routers.grafana-https.entrypoints=https"
      - "traefik.http.routers.grafana-https.rule=Host(`grafana.oiba.de`)"
      - "traefik.http.routers.grafana-https.tls.certresolver=acme"
      - "traefik.http.routers.grafana-https.middlewares=private-chain@file"
      - "traefik.http.routers.grafana-https.service=grafana-https"
      - "traefik.http.services.grafana-https.loadbalancer.server.port=3000"
      - "com.centurylinklabs.watchtower.enable=true"
networks:
  monitoring:
    external: true
```

Figure 13: Docker Compose Configuration for Deploying Grafana (Own Illustration).

```yaml
version: '3.8'
services:
  portainer:
    image: portainer/portainer-ee:latest
    container_name: portainer
    hostname: portainer
    restart: unless-stopped
    security_opt:
      - no-new-privileges
    env_file:
      - ./portainer.env
      - ../common.env
    networks:
      - portainer
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ./data:/data
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=portainer"
      - "traefik.http.routers.portainer.entrypoints=http"
      - "traefik.http.routers.portainer.rule=Host(`docker.oiba.de`)"
      - "traefik.http.routers.portainer.middlewares=https-redirect@file"
      - "traefik.http.routers.portainer.service=portainer-https"
      - "traefik.http.routers.portainer-https.entrypoints=https"
      - "traefik.http.routers.portainer-https.rule=Host(`docker.oiba.de`)"
      - "traefik.http.routers.portainer-https.tls.certresolver=acme"
      - "traefik.http.routers.portainer-https.middlewares=private-chain@file"
      - "traefik.http.routers.portainer-https.service=portainer-https"
      - "traefik.http.services.portainer-https.loadbalancer.server.port=9000"
      - "com.centurylinklabs.watchtower.enable=true"
networks:
  portainer:
    external: true
```

Figure 14: Docker Compose Configuration for Deploying Portainer (Own Illustration).

6.2.2 Selecting the Solution Approach

Both Ansible and Terraform have native plugins that support the deployment of Docker containers. These tools enable container orchestration by leveraging the Docker run command and defining containers through their declarative syntax. However, the expectation that directly utilizing these tools would streamline the deployment process and address the redundancy issues observed in the Docker Compose files was not met. Instead, the case study's environment presented a new set of challenges when using Ansible and Terraform directly, which ultimately led to the exploration of an alternative approach.

One approach to managing container stacks is using docker-compose.yml files, which are generated by leveraging Ansible's Jinja2 templating engine. Ansible utilizes Jinja2 templates to define variables that are substituted with controllable values during runtime. This feature would enable the creation of customizable docker-compose.yml files that can adapt to different configurations in various environments or deployments (Been et al., 2022, pp. 11–15).

Although templating the docker-compose.yml files introduced additional complexity compared to using the already existing ones, the decision was justified. This approach resolved the issues explained in the previous subsection, resulting in more streamlined and maintainable files. Additionally, the use of templating allowed for greater flexibility and control over the configuration process.

To effectively implement this solution, a docker-compose.yml template was crafted in Ansible, designed to incorporate all requisite settings for deploying all 67 services. Each service possesses unique configurations that require populating the template with service-specific variables. The intricate details of that implementation will be discussed in the following section.

6.2.3 Technical Documentation of the Docker Compose Template

Given the extensive nature of the code, which comprises 170 lines, providing a detailed elaboration on the functioning of the Docker Compose template within the confines of this thesis is impractical. Instead, the template's essence lies in its ability to specify multiple default values for recurring entries. This is achieved by using for-loops and if-conditions to selectively include the necessary variables, resulting in a final Docker Compose file that is both concise and customized. The Jinja2 template, which can be identified by the .j2 file extension, is located at "/workspace/homelab/roles/0docker_service/templates/docker-compose.yml.j2".

The process by which the template is populated with the necessary variables is crucial to the overall process. Ansible Playbooks offer flexibility by enabling execution on individual hosts or groups of hosts, performing specific tasks, or applying roles. Ansible roles simplify

playbook complexity by grouping tasks, templates, and variables specific to a server function, making it easier to configure services across diverse hosts with precision (Geerling, 2020, VIII–IX; Wurster et al., 2020, pp. 66–67, 2020, p. 67, 2020, p. 70).

Table 4 illustrates that multiple instances of a Docker service run independently on different Docker hosts, each with its own configuration. When creating the Docker service, it is important to write it in a generic manner that allows for customization to adapt to the specific needs of the Docker hosts within the context of the case study's environment. Furthermore, the objective was to create a detailed file for each host that outlines all deployed services, providing a quick and easy overview of the host's operations. It is also indispensable to have the ability to add and remove services dynamically and efficiently, which was a key consideration.

To accomplish this, a playbook was created for each host, with the relevant roles assigned. This enables service-specific task execution, configuration file processing, and variable incorporation. Each Docker service is contained within a single role. For better clarity the contents of the final Traefik role are displayed in Figure 15.

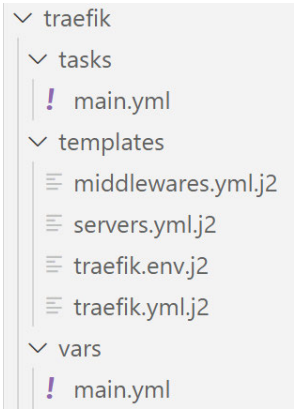


Figure 15: Structure and Components of the Traefik Role (Own Illustration).

However, that approach would still require placing the Docker Compose template in each Docker service's role folder. To further streamline the process, a wrapper role called “0docker_service” was created. This role contains the Docker Compose template in its template folder and cyclically processes each selected Docker service role folder defined in the “docker_service_roles” variable. In other words, the role "0docker_service" nests inside itself other roles which are controlled through the “docker_service_roles” variable and Figure 16 provides a rudimentary example of it.

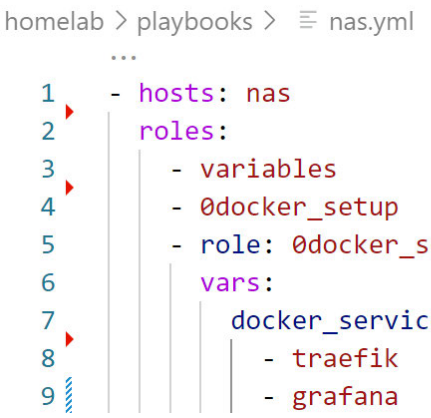


Figure 16: Example Playbook Demonstrating the Usage of the 0docker_service Role (Own Illustration).

As previously stated, this IaC project heavily relies on the use of variables to manage configurations across various hosts for the deployment of Docker services. The first task in the main.yml file of the 0docker_service role, shown in Figure 17, is crucial for managing these variables.

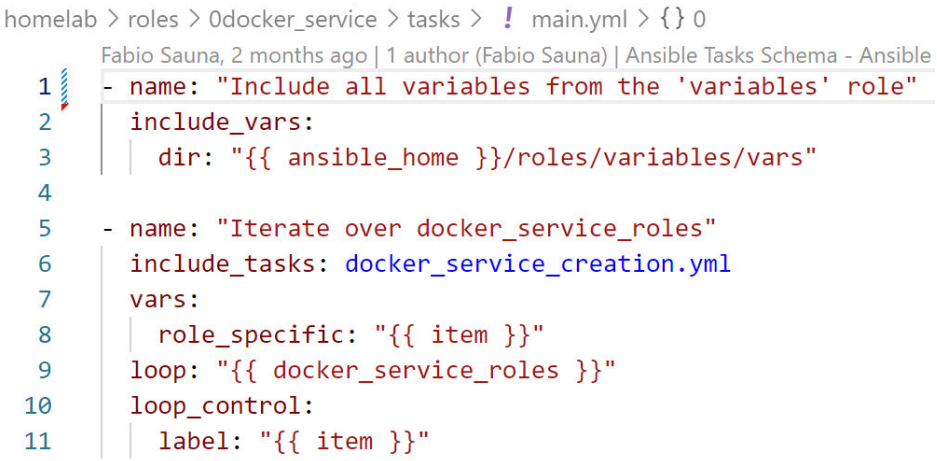


Figure 17: Contents of the Tasks for the 0docker_service Role (Own Illustration).

This task serves as a conduit by loading the variables from the designated variable role, which is central to the functioning of the entire IaC project. They contain all critical variables, which are categorized as host-specific and general-purpose.

Ansible automatically matches host-specific variables by leveraging the correlation between the target hostname and the filename of the variables files (Hochstein and Moser, 2017, pp. 2–8). These variables are crucial for adapting to the specific needs of the host. For instance, Docker networks that must be established on the host before deploying Docker services can be defined there. Without these networks, deployment would fail, like a Docker host without Docker installed. The IaC project has addressed the steps for automating Docker installation and initializing Docker networks, but a detailed explanation is beyond the scope of this discussion due to space limitations.

In the main.yml file, general-purpose variables are used to provide essential information for the creation of Docker services. These variables dictate the paths where docker-compose and template files will be located on the host machine, as well as the user and group IDs and file permissions.

The second task, titled “Iterate over docker_service_roles”, loops through each role defined in the docker_service_roles variable and executes the tasks inside docker_service_creation.yml. These subtasks encompass all the necessary steps for deploying a Docker service. Figure 18 illustrates a representative selection of these subtasks.

homelab > roles > 0docker_service > tasks > ! docker_service_creation.yml > {} 7 > abc when

You, 9 hours ago | 2 authors (Fabio Sauna and others) | Ansible Tasks Schema - Ansible tasks file (ansible.json)

```yaml
- name: "[{{ role_specific }}] Include role-specific vars"
  include_vars: "{{ ansible_home }}/roles/{{ role_specific }}/vars/main.yml"

- name: "[{{ role_specific }}] Create role directory"
  become: true
  file:
    path: "{{ docker_base_path }}/{{ docker_role_name }}"
    state: directory
    owner: "{{ common_username }}"
    group: "{{ common_group_gid }}"
    mode: "{{ default_file_permission_mode }}"

- name: "[{{ role_specific }}] Template Docker Compose file"
  template:
    src: "{{ ansible_home }}/roles/0docker_service/templates/docker-compose.yml.j2"
    dest: "{{ docker_base_path }}/{{ docker_role_name }}/docker-compose.yml"
    owner: "{{ common_username }}"
    group: "{{ common_group_gid }}"
    mode: "{{ default_file_permission_mode }}"
  become: true
```

Figure 18: Excerpt from the Subtasks within the docker_service_creation.yml File (Own Illustration).

The initial subtasks load the variables specific to the Docker service. This fills the Docker Compose template with the desired configurations, as shown for the Grafana service in Figure 19. After completing the loading phase, the process continues with creating the required directories, templating the configuration files, and executing the docker-compose command to create the service. It is important to note that variables are used to parameterize the execution of most subtasks but detailing them is beyond the scope of this thesis.

The wrapper approach underwent several revisions to achieve its current streamlined process. In the early iterations, each role had its specific task folder, where the logic now housed in docker_service_creation.yml was originally located. This scattered arrangement required that any single change made to the tasks would have to be painstakingly duplicated for every individual role, a process that was both error-prone and time-consuming. As the system expanded with the addition of new roles, it became obvious that this method would

not scale well. To streamline the process, the tasks' folder for each role was further simplified to only include a task-loader that would only load a centrally located docker_service_creation.yml file. However, even with that change, it was still necessary to minimize duplicated code, as with the second iteration a separate tasks folder was required for each docker service solely to contain the task-loader code.

The refinement process ultimately resulted in the adoption of the current wrapper approach. This method requires only the vars folder for the Docker service to populate the main.yml file with all the essential variables needed by the Docker Compose template. Therefore, there is no more duplicated code, and configuration files or service-specific tasks are optional and can be added if the service requires them.

To illustrate the effectiveness of the wrapper approach, the DMS paperless-ngx stack was implemented using this method. The stack requires five containers to function properly, including an OCR and a MariaDB container. By inputting only the necessary service variables, the original docker-compose.yml was reduced from 94 to 59 lines of code in the main.yml file. This demonstrates a significant reduction of approximately 37%. The comparison between the docker-compose.yml for Grafana in Figure 13 and the new main.yml in Figure 19 highlights the simplicity and reduced code complexity of the wrapper approach. The configuration is shorter and more straightforward, resulting in the completion of the intended objective.

Figure 19: Sample main.yml File for Grafana Role to Populate the Docker Compose Template (Own Illustration).

6.3 Objective 2: Centralized Service URL Management

The next milestone in configuration management is the centralization of all service URLs. In the Grafana deployment seen in Figure 19 line 18, the URL is specified by a variable that, during the Ansible execution, is dynamically replaced with the actual URL “grafana.oiba.de”. This section explains and demonstrates the substitution process and how it was integrated with the existing monitoring systems.

The centralized URL management system is structured around the variable named “endpoints”, which is defined inside the variable role. This variable is integral to the system's configuration scheme, as it contains references to all the associated URLs. Thanks to its highly adaptable design, the endpoints variable can accommodate a diverse array of service configurations without any hassle.

The construction of the system begins by assigning a unique identifier to each service in the endpoints variable. These identifiers can then be accessed by calling “endpoints.name”. To avoid hard-coding the name, it can be dynamically retrieved and integrated using Ansible. Specifically, substituting “endpoints.name” with “endpoints[services[0].name]” introduces such a dynamic placeholder. The placeholder “[services[0].name]” is then automatically replaced with the actual name of the first service listed under the “services” in the variables of the specific Docker service.

After obtaining the service name, Ansible replaces the second placeholder, “[ansible_facts['hostname']]”, with the hostname of the target. It then retrieves the URL value specified in the endpoints variable for that host. By matching the URL with its designated hostname, the system ensures that the correct URL is consistently associated with the appropriate Docker host.

To clarify how the variable substitution operates, let's take the Grafana deployment as an instance:

“`{{endpoints[services[0].name][ansible_facts['hostname']]}}`”

The expression above uses placeholder elements to produce the URL “grafana.oiba.de”. Initially, Ansible aligns the “Grafana” service name from the “services” list, as outlined in Figure 19 line 4, with its equivalent in the endpoints variable. A glimpse of the endpoints variable is provided in Figure 20, where the “Grafana” service is mapped, pinpointing the selection of URLs within “endpoints.grafana” (lines 223-224). Next, the system identifies the correct host by its hostname, recognized here as “nas”, which consequently leads to the derivation of the URL “grafana.oiba.de”.

Figure 20: Structure of the Endpoints Variable for Dynamic URL Construction (Own Illustration).

As seen in Figure 20 the architecture of the endpoints variable is dynamic, enabling URL construction based on the service name and current hostname. It also accommodates both single and multiple entries per service and host, providing significant customization and flexibility, which is utilized throughout the entire IaC project to populate various configurations with precise URLs.

6.3.1 Utilization in Monitoring Systems

One area where the centralized URL management is particularly necessary is when accurately measuring service availability and performance. This requires using the correct URLs for monitoring systems and ensuring proper configuration of these tools. In the case study, Uptime Kuma and Prometheus turned out to be invaluable assets for tracking service health.

Prometheus exporters provide metrics about services through a URL, which the Prometheus server scrapes to monitor service health (Prometheus, 2024). The configuration of Prometheus must include these URLs, and when templating its configuration, it dynamically incorporates services marked with "`prometheus_exporter: true`" in the endpoints variable. It should be noted that this thesis cannot provide detailed coverage of the entire Prometheus template. In summary, Ansible iterates over each Prometheus-enabled endpoint and applies the specified settings, including additional parameters, custom query parameters, authentication credentials, and relabeling configurations, to generate a fully functional and tailored Prometheus configuration.

Complementing Prometheus, Uptime Kuma checks the status of services to determine whether they are operational (louislam/uptime-kuma, 2024). The uptimekuma.yml Playbook utilizes a plugin to interact with the Uptime Kuma API and set up individual monitors for various services' monitoring configurations. To maintain conciseness, a detailed explanation of the 167-line playbook is beyond the scope of this thesis. However, Figure 21 displays a code snippet that adds the services URLs to be monitored. Like Prometheus, these URLs are taken from the endpoints variable. Further customization through the endpoints variable has been added to enhance functionality, such as the ability to ignore a defined URL and to monitor the status of Docker containers, for instance "unifi_mariadb", which as a database container does not have a URL to probe.

```
40      - name: Generate the monitor list from the endpoints
41        set_fact:
42          defined_monitor_list: >-
60            {% for service, details in endpoints.items() %}
61              {% if details.uptimekuma_monitoring is not defined or details.uptimekuma_monitoring != false %}
62                {% for host, url in details.items() if details is mapping %}
63                  {% if host in endpoint_hosts %}
64                    {% if url is string %}
65                      {"name": "{{ service | capitalize }} HTTP ({{ host | upper }})", "parent_host": "{{ host | upper }}"
66                    {% elif url is iterable %}
67                      {% for u in url %}
68                        {"name": "{{ service | capitalize }}{% if loop.index > 1 %} {{ loop.index }}{% endif %} HTTP ({{ h
69                      {% endfor %}
70                    {% endif %}
71                  {% endif %}
72                {% endfor %}
73                {% if details.uptimekuma_container is defined %}
74                  {% for container, container_details in details.uptimekuma_container.items() %}
75                    {"name": "{{ container | capitalize }} CONTAINER ({{ container_details.docker_connection_host | upper
76                  {% endfor %}
77                {% endif %}
78              {% endif %}
79            {% endfor %}
80          ]
```

Figure 21: Excerpt from the Uptime Kuma Playbook Displaying the Integration of Service URLs (Own Illustration).

The endpoints variable has streamlined the deployment and monitoring processes, ensuring that services such as Prometheus and Uptime Kuma receive accurate URLs for optimal operation. This dynamic approach to URL management highlights the system's flexibility, enhancing automation and monitoring efficiency.

6.4 Objective 3: Automatic Deployment and Management of IT Infrastructure

Infrastructure provisioning is a crucial step in establishing an automated IT environment. Terraform enables the specification of infrastructure as code, which can be consistently and repeatedly applied to create and manage infrastructure. This process involves allocating and configuring the necessary (virtual) hardware on which all services will operate.

The case study's infrastructure is managed and defined using Terraform, which interfaces with both the Proxmox hypervisor and Microsoft Azure cloud services. Figure 22 illustrates the structure of the Terraform configuration folder, highlighting two primary files critical for infrastructure creation: azure.tf, which contains all Azure-related infrastructure code, and main.tf, which details the configurations specific to Proxmox.

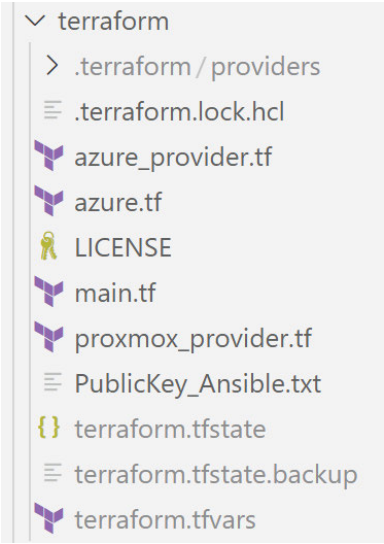


Figure 22: Overview of the Terraform Configuration Directory Structure (Own Illustration).

When executing the Terraform configuration files, a Terraform state file is generated. This state file serves as a source of truth, capturing the precise configuration and current state of the provisioned infrastructure. Terraform uses this state to determine the necessary actions to align real-world resources with the configuration defined in code, enabling precise tracking and management of deployed resources (Brikman, 2022, pp. 37–38, 2022, p. 6, 2022, p. 407; Wurster et al., 2020, pp. 66–67, 2020, p. 70).

To streamline variable definitions and minimize redundancy, the terraform.tfvars file is used. The variables encompass crucial information, such as the Proxmox URL and login credentials, along with Azure configurations, including the admin username, password, and resource locations.

The Proxmox provider for Terraform enables the creation of virtual machines and Linux Containers through various methods, such as attaching a boot ISO for OS installation, cloning from a pre-configured template, or importing existing infrastructure. However, for this case study, creating resources from scratch was deemed unnecessary as it does not align

with the project's objectives. Instead, the focus was on importing the existing infrastructure into Terraform for further configuration.

The main.tf file defines the full infrastructure, and Figure 23 provides an example of the Proxmox Terraform code. The figure details the configuration of the Proxmox Backup Server VM, including various parameters such as VM settings, boot options, OS specifications, CPU and RAM configuration, and disk setup. The other infrastructure is similarly defined and does not require further details.

```
resource "proxmox_vm_qemu" "bak" {
  name              = "BAK"
  vmid              = 110
  target_node       = "prox"
  bios              = "ovmf"
  machine           = "q35"
  vm_state          = "running"
  clone             = "BAKTEMPLATE"
  onboot            = true
  automatic_reboot  = true
  startup           = "order=7"
  qemu_os           = "l26"
  os_type           = "6.x - 2.6 Kernel"
  agent             = 1
  sockets           = 1
  cores             = 2
  cpu               = "x86-64-v3"
  memory            = 4096
  scsihw = "virtio-scsi-pci"
  disks {
    scsi {
      scsi0 {
        disk {
          size = 32
          storage = "local-lvm"
        }
      }
    }
  }
  efidisk {
    efitype = "4m"
    storage = "local-lvm"
  }
  network {
    model     = "virtio"
    bridge    = "vmbr0"
    macaddr   = "00:0C:29:7B:40:A8"
    tag       = 100
  }
}
```

Figure 23: Sample Terraform Code: Proxmox Backup Server Configuration (Own Illustration).

The Azure setup followed a similar sequence of infrastructure definition. The setup process for Azure involves several discrete steps. These include creating a resource group, a virtual network, and a subnet, followed by allocating a public IP address. These steps are necessary

before selecting the virtual machine type, such as “Standard_E2s_v5”, and the OS disk. In this case study, Ubuntu 22 LTS was used as the base image. Additionally, the configuration includes passing over the public key for the Ansible user to facilitate remote management of the machine. The Azure code will not be discussed in detail due to its similarity to the main.tf used for Proxmox configurations.

By defining and managing infrastructure through Terraform, the case study demonstrates a robust and repeatable process for deploying infrastructure in both on-premises and cloud-based environments.

7. Demonstration of the Research Findings

This chapter presents the research findings in a clear and comprehensive manner, analyzing the attained results. It serves a dual purpose by illustrating the outcomes and aligning with Peffers' DSRM, specifically fulfilling the Demonstration step. The Design and Development step required detailing the source code functioning, partly interweaving with the Demonstration step because it would show the artifact's capabilities in action. However, this chapter shifts focus from code intricacies to an evaluation of the research findings.

The case study contained a total of 80 services, of which 67 were Docker-based. Consequently, the initial goal of the research was to streamline the deployment of Docker services by employing a standardized Docker Compose template. Owing to time constraints, it was not feasible to deploy all 67 of them. Out of those, 33 services were successfully deployed, as detailed in Table 2 in Section 4.2 and Table 4 in the Appendix, with the “Status” column denoting the deployment status of each service. The objective was to create a template that can deploy all Docker services. The fact that almost half of the services are fully provisioned and configured in IaC demonstrates the capability and effectiveness of the Docker Compose template. These services have remained operational for over six weeks. Their performance was assessed both manually and through the case study's monitoring system, which verified their operational integrity.

The second objective was to centralize URL management to enable dynamic and instantaneous updating of monitoring targets, a critical feature needed due to the rapid obsolescence of such settings. The centralized system contains the URLs of all services deployed within the case study. It has been fully operational for over a month without any malfunctions. A significant accomplishment is the elimination of duplicated hard-coded URLs. The codebase once defines URLs as variables and references them throughout the configuration and monitoring system.

The third objective aimed to address the challenge of snowflake systems and the need for automated provisioning infrastructure by utilizing Terraform. Due to time limitations, the objective was not to achieve complete automation, but to manage the entire infrastructure using Terraform. By defining each VM and LXC in Terraform code, this objective was successfully accomplished. Codifying and managing infrastructure are a crucial step towards scalable and reproducible deployment processes. It establishes a solid foundation for future efforts to fully automate infrastructure provisioning.

The progress achieved in all three objectives has resulted in a tangible artifact, of which the full source code is available in a public repository hosted on GitHub at: https://github.com/fabsau/workspace.

To ensure the integrity and transparency of the data presented in this thesis, a dedicated branch named “ba-submission” has been created to demarcate the code's status at the time of thesis submission. It should be noted that further code development will occur after submission.

To quantitatively analyze the contents of the repository, an assessment was conducted using Tokei, a tool that generates metrics of codebases. These metrics include file count, total lines, and the distribution of these lines into code, comments, and blanks for each programming language (GitHub, 2024b). The complete statistics are summarized in Figure 24.

Language	Files	Lines	Code	Comments	Blanks
HCL	4	651	456	82	113
INI	1	60	43	0	17
Shell	1	119	85	14	20
Plain Text	1	1	0	1	0
YAML	67	2925	2584	163	178
Total	74	3756	3168	260	328

Figure 24: Codebase Statistics Overview (Own Illustration).

The repository contains primarily Ansible code, which makes up most of the repository, with 3.168 lines of code. Approximately 81% or 2.584 lines are dedicated to Ansible scripts. Additionally, there is Terraform code (HCL) comprising 456 lines. The remaining code consists of a single INI file, specifically the 43-line-long Ansible inventory file, and 85 lines of Shell scripts, which are outside the scope of this thesis.

8. Evaluation and Discussion of the Results

This chapter evaluates the results of the research methodology and design used to address the research question, following the Evaluation step guidelines of DSRM. The assessment determines the extent to which the created artifact fulfills the objectives outlined at the beginning of the research.

The first stage of the thesis involved conducting a thorough literature review. The review revealed that, at the time of writing, there were no complete examples of infrastructure deployment using IaC, with the code being entirely open-sourced. To support this observation, a survey was conducted, asking industry professionals to evaluate the existence of this gap. The survey results appear to confirm this assumption.

Critiquing the survey methodology is appropriate. Distributing the survey through an online Reddit community may have introduced bias, as the responses cannot be independently verified. Additionally, the anonymity of the responses might lead to misunderstandings due to the lack of direct interaction between the researcher and the participants. However, when searching for extensive IaC source code examples, the search yields no results. This underscores the initial observation of a gap in both the literature and industry practices.

The main objective of this research was to create a source code, referred to as an artifact. The study utilized the Design Science Research Methodology by Peffers to guide the research process. Although some may argue that DSRM was unnecessary since it resulted in a traditional thesis format, its adoption effectively refined the research scope, transforming a vague research question into a well-defined methodological sequence and decisively shaping the development process.

To develop a practical and usable artifact, it was crucial to create a well-defined environment in which to implement it. Therefore, a case study focused on a Homelab was undertaken. Expert interviews were conducted to substantiate the Homelab as a fitting choice. These discussions contributed to the development of an IT infrastructure function map, which was instrumental in pinpointing the commonly shared functions of IT infrastructure.

Although the three interviewees had a combined experience of over 55 years, the research's transparency was slightly compromised by the absence of interview transcriptions. This omission was due to the constraints of time and space within the thesis. Additionally, including a larger pool of interview participants could have improved the findings' objectivity.

Although some may argue for a more extensive infrastructure environment or multiple case studies for a more comprehensive study, the depth of insight gained from a single, focused case study should not be underestimated. Flyvbjerg (2006) argues that practical knowledge from an individual case can yield valuable insights relevant to research questions (Flyvbjerg, 2006, pp. 223–224). After completing the mapping exercise that compared the common

functions of SME infrastructures with those of the selected Homelab, it can be confirmed that the chosen case study environment was appropriate for the development of the artifact.
That required the establishment of clear objectives. To define them, a comprehensive literature review was undertaken, which included recent academic papers, books, and industry reports. This review provided a solid foundation for creating the artifact and assessing the practical implementation of Infrastructure as Code. The objectives were derived from the most desired features identified in the survey responses and adapted to be feasible within the case study.
The methods for achieving the objectives have been outlined in the "Design and Development" and "Demonstration" steps and were descriptively presented within the case study, proving to be viable. However, the "Evaluation" step primarily uses a descriptive approach, which can make the interpretation of results subjective. The SMART criteria provide an objective means of evaluating the results. However, to further strengthen the evaluation, additional research methods could have been considered, such as incorporating external code reviews. Unfortunately, due to space and time constraints, this was not possible.
The evaluation of the SMART criteria for the first objective shows that all services migrated using the template are operational. The remaining services can also be migrated using this template. However, the template does not support less commonly used Docker Compose file settings such as "ulimits", "tmpfs", and "pid" (Docker Documentation, 2024c). However, the template's open-source nature permits practitioners to continually modify and enhance it.
Upon inspecting the repository's content, the second objective's criteria is met, as there are no duplicated hard-coded URLs. The third objective was to incorporate a hybrid cloud infrastructure into Terraform, and while the criteria are met the scope was limited. A proper Infrastructure as Code approach involves complete automation, from machine provisioning to service deployment and configuration. Due to time constraints, it was not feasible to recreate each virtual machine from scratch and manually reconfigure it using Terraform and Ansible.

Determining whether the artifact effectively addresses the research question can also depend on the final step of the DSRM framework: communication. The impact of the artifact, once published and disseminated, will also serve as a litmus test, validating the six weeks of coding by a bachelor's degree student. It will demonstrate whether the artifact has established a foundational basis for further development that provides practical value to SMEs. This raises doubts about the effectiveness of reaching SMEs, but efforts will be made to overcome this limitation.
The researcher will aim to maximize the potential of the available dissemination opportunities, despite being constraint with limited channels.

9. Conclusion

The central thesis of this study asserts the transformative impact of cloud computing and the role of Infrastructure as Code in efficiently managing IT environments for SMEs, particularly in mitigating the challenges associated with manual configuration and monolithic application structures. The study aimed to demonstrate an exemplary IaC deployment that addresses the needs of SMEs and fills the significant knowledge gap in the existing literature by providing an actionable and publicly available IaC artifact.

The study identified needs within an SME-relevant IT infrastructure case and successfully derived objectives. An artifact was created using Ansible and Terraform to prove the feasibility of achieving those objectives. The artifact showcased a templating mechanism for Docker services, centralized URL management for service configuration and monitoring, and the integration of a hybrid cloud infrastructure with Terraform.

This research contributes to the academic and practical understanding of IaC. It provides an example of IaC deployment that fills the need for comprehensive code examples and serves as a platform for further exploration and innovation in this area. This example provides a reference model that can be adapted and extended, potentially reducing the learning curve and implementation barriers associated with IaC adoption in SMEs.

Although the research has potential value, it encountered specific constraints. The dissemination was limited to a few channels, which may have hindered the study's exposure to a broader audience, including industry experts and academics who could have conducted a code review. Such a review process could have offered further insights and confirmed the artifact's robustness by incorporating feedback from established authorities in the field.

Furthermore, the study was limited by the scope of automation achieved in server provisioning, which did not accomplish complete automation. This limitation may not fully meet the needs of SMEs aiming for a wholly automated IaC solution.

Despite these limitations, this research concludes that this study has advanced the body of knowledge on IaC in the context of SMEs by filling a gap in practical examples. It has also laid the groundwork for future efforts to refine and extend automated infrastructure management practices.

9.1 Future Research Recommendations

Future studies should also aim to develop comprehensive solutions tailored to real-world IT challenges and open-source their results. By analyzing various case studies and making their code fully accessible, they could facilitate collaborative efforts to generate knowledge that considers other aspects not covered in this study. This could help close the current knowledge gap and improve the Infrastructure as Code domain.

List of References

A. Rahman, Effat Farhana, Chris Parnin and L. Williams (2020) 'Gang of Eight: A Defect Taxonomy for Infrastructure as Code Scripts', *International Conference on Software Engineering* [Online]. Available at https://www.semanticscholar.org/paper/Gang-of-Eight%3A-A-Defect-Taxonomy-for-Infrastructure-Rahman-Farhana/4078526b6ac5ad94b2aa55061d24648481682c2b.

Abts, D. and Mülder, W. (2017) *Grundkurs Wirtschaftsinformatik: Eine kompakte und praxisorientierte Einführung*, 9th edn, Wiesbaden, Springer Fachmedien Wiesbaden; Imprint: Springer Vieweg.

Alshamaila, Y., Papagiannidis, S. and Li, F. (2013) 'Cloud computing adoption by SMEs in the north east of England', *Journal of Enterprise Information Management*, vol. 26, no. 3, pp. 250–275 [Online]. DOI: 10.1108/17410391311325225.

Ambasna-jones, M. (2018) 'Automation key to unravelling mysteries of the universe at CERN', *ComputerWeekly.com*, 12 April [Online]. Available at https://www.computerweekly.com/feature/Automation-key-to-unravelling-mysteries-of-the-universe-at-CERN (Accessed 1 March 2024).

Artac, M., Borovsak, T., Nitto, E. D., Guerriero, M. and Tamburri, D. (2017) 'DevOps: Introducing Infrastructure-as-Code', *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 497–498.

Arul Elumalai, James Kaplan, Mike Newborn and Roger Roberts (2018) *Making a secure transition to the public cloud,* McKinsey.

Azure | Terraform | HashiCorp Developer (2024) *Azure | Terraform | HashiCorp Developer* [Online]. Available at https://developer.hashicorp.com/terraform/tutorials/azure-get-started (Accessed 29 March 2024).

Balalaie, A., Heydarnoori, A. and Jamshidi, P. (2016) 'Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture', *IEEE Software*, vol. 33, no. 3, pp. 42–52.

Bass, L., Weber, I. M. and Zhu, L. (2015) *DevOps: A software architect's perspective*, New York, Addison-Wesley Professional.

Been, H., Keilholz, E. and Staal, E. (2022) *Azure infrastructure as code : with ARM templates and BICEP*, Shelter Island, NY, Manning Publications Co.

Bourgeois, D. T., Smith, J. L., Wang, S. and Mortati, J. (2019) 'Information Systems for Business and Beyond', *Open Textbooks* [Online]. Available at https://login.ezproxy-dhma.redi-bw.de/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=edsbas&AN=edsbas.4081E5BA&lang=de&site=eds-live.

Bovend'Eerdt, T. J. H., Botell, R. E. and Wade, D. T. (2009) 'Writing SMART rehabilitation goals and achieving goal attainment scaling: a practical guide', *Clinical rehabilitation*, vol. 23, no. 4, pp. 352–361.

Brikman, Y. (2022) *Terraform up & running : writing infrastructure as code*, [Sebastopol, California], O'Reilly Media, Inc.

Burgess, M. (1995) 'A Site Configuration Engine', *Computing Systems* [Online]. Available at https://www.semanticscholar.org/paper/A-Site-Configuration-Engine-Burgess/abc44de8d6f4c2f165a01c147637b31e63e42ead.

C. David Hylender, Philippe Langlois, Alex Pinto and Suzanne Widup (2023) *Verizon 2023 Data Breach Investigations Report*.

Campbell, B. (2020) *The Definitive guide to AWS infrastructure automation: Craft infrastructure-as-code solutions*, Berkeley, CA, Apress.

Chen, L. (2017) 'Continuous Delivery: Overcoming adoption challenges', *Journal of Systems and Software*, vol. 128, pp. 72–86 [Online]. DOI: 10.1016/j.jss.2017.02.013.

Chen, W., Wu, G. and Wei, J. (2018) 'An Approach to Identifying Error Patterns for Infrastructure as Code', *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW).* Memphis, TN, 10/15/2018 - 10/18/2018. [Place of publication not identified], IEEE, pp. 124–129.

Christian Endres, Uwe Breitenbücher, Michael Falkenthal, Oliver Kopp, F. Leymann and Johannes Wettinger (2017) 'Declarative vs . Imperative : Two Modeling Patterns for the Automated Deployment of Applications' [Online]. Available at https://www.semanticscholar.org/paper/Declarative-vs-.-Imperative-%3A-Two-Modeling-Patterns-Endres-Breitenb%C3%BCcher/8ae91607dc60991950b08b5b72a20893878b8907.

Cloud Native Computing Foundation (2023) *Gitops Microsurvey: Learning on the job as GitOps goes mainstream*.

Creswell, J. W. (2009) *Research design: Qualitative, quantitative, and mixed methods approaches*, 3rd edn, Thousand Oaks Calif., SAGE Publications.

Creswell, J. W. and Plano Clark, V. L. (2018) *Designing and conducting mixed methods research*, Los Angeles, SAGE.

D. Sokolowski and G. Salvaneschi (2023) 'Towards Reliable Infrastructure as Code', *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pp. 318–321.

Dalla Palma, S., Di Nucci, D., Palomba, F. and Tamburri, D. A. (2020) 'Toward a catalog of software quality metrics for infrastructure code', *Journal of Systems and Software*, vol. 170, p. 110726 [Online]. DOI: 10.1016/j.jss.2020.110726.

Dan London (2014) *NASA: INCREASING CLOUD EFFICIENCY WITH ANSIBLE AND ANSIBLE TOWER,* Ansible [Online]. Available at https://www.ansible.com/blog/nasa-automation.

Demir, G. K. (2023) 'Homelab Part 1: Why to Build a Homelab, and Who is it for?', *Turk Telekom Bulut Teknolojileri*, 12 November [Online]. Available at https://medium.com/t%C3%BCrk-telekom-bulut-teknolojileri/homelab-part-1-why-to-build-a-homelab-and-who-is-it-for-d54f59b9e560 (Accessed 20 March 2024).

Derek DeBellis, Amanda Lewis and Daniella Villalba (2023) *State of DevOps Report 2023*.

Docker Documentation (2024a) *Docker Compose overview* [Online]. Available at https://docs.docker.com/compose/ (Accessed 30 March 2024).

Docker Documentation (2024b) *Networking overview* [Online]. Available at https://docs.docker.com/network/ (Accessed 30 March 2024).

Docker Documentation (2024c) *Compose file version 3 reference* [Online]. Available at https://docs.docker.com/compose/compose-file/compose-file-v3/ (Accessed 1 April 2024).

Erl, T., Puttini, R. and Mahmood, Z. (2013) *Cloud computing: Concepts, technology, & architecture / Thomas Erl, Zaigham Mahmood and Richardo Puttini*, Upper Saddle River, NJ, Prentice Hall.

F. Beetz and Simon Harrer (2022) 'GitOps: The Evolution of DevOps?', *IEEE Software* [Online]. Available at https://www.semanticscholar.org/paper/GitOps%3A-The-Evolution-of-DevOps-Beetz-Harrer/60ed7e86e2bef4997f031f9efdece921c1a9449c.

Filkins, B. (2018) 'IT Security Spending Trends' [Online]. Available at https://www.semanticscholar.org/paper/IT-Security-Spending-Trends-Filkins-Hardy/c56c3a093730e049131e55143ccf03698ca6bb66.

Flexera (2023) *2023 State of the Cloud Report*.

Flexera Software (2024) *Usage of cloud configuration tools worldwide in 2023, current and planned* [Online]. Available at https://www.statista.com/statistics/511293/worldwide-survey-cloud-devops-tools/ (Accessed 10 March 2024).

Flyvbjerg, B. (2006) 'Five Misunderstandings About Case-Study Research', *Qualitative Inquiry* [Online]. Available at https://www.semanticscholar.org/paper/Five-Misunderstandings-About-Case-Study-Research-Flyvbjerg/5adfa92966999563cfa5e83b4c74f3e730bd3d38.

Forrester Consulting (2022) *HashiCorp 2022 State of Cloud Strategy Survey*.

Fortune Business Insights (2023) *Infrastructure as Code Market Size, Share | Growth Report, 2030* [Online]. Available at https://www.fortunebusinessinsights.com/infrastructure-as-code-market-108777 (Accessed 2 March 2024).

Fowler, M. and Beck, K. (1999) *Refactoring: Improving the design of existing code*, Reading MA, Addison-Wesley.

Fryman, J. (2014) *DNS Outage Post Mortem* [Online]. Available at https://github.blog/2014-01-18-dns-outage-post-mortem/ (Accessed 1 March 2024).

Gaba, I. (2020) 'What is Continuous Integration, Deployment, and Delivery?', *Simplilearn*, 9 July [Online]. Available at https://www.simplilearn.com/tutorials/devops-tutorial/continuous-delivery-and-continuous-deployment (Accessed 22 March 2024).

Geerling, J. (2020) *Ansible for DevOps: Server and configuration management for humans* [Online], 2nd edn, [Victoria, British Columbia], Leanpub. Available at http://www.ansiblefordevops.com/.

GitHub (2024a) *Telmate/terraform-provider-proxmox: Terraform provider plugin for proxmox* [Online]. Available at https://github.com/Telmate/terraform-provider-proxmox/releases (Accessed 29 March 2024).

GitHub (2024b) *XAMPPRocky/tokei: Count your code, quickly* [Online]. Available at https://github.com/XAMPPRocky/tokei (Accessed 1 April 2024).

Google Trends (2024) *Google Trends* [Online]. Available at https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0b6sp%2C%2Fm%2F0c3tq11%2CIaC&hl=en-GB (Accessed 12 March 2024).

Greg Watts and Norman Watts (2018) 'Goal setting', *CPD in the Built Environment* [Online]. Available at https://www.semanticscholar.org/paper/Goal-setting-Watts-Watts/5c17d97271ad60d4491e16df51eeee21cdd4811d.

Guerriero, M., Garriga, M., Tamburri, D. A. and Palomba, F. (2019) 'Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry', *IEEE International Conference on Software Maintenance and Evolution*, pp. 580–589 [Online]. DOI: 10.1109/ICSME.2019.00092.

Håvard Myrbakken and Ricardo Colomo Palacios (2017) 'DevSecOps: A Multivocal Literature Review', *International Conference on Software Process Improvement and Capability Determination* [Online]. Available at https://www.semanticscholar.org/paper/DevSecOps%3A-A-Multivocal-Literature-Review-Myrbakken-Palacios/6487ea3275f532b50b55d6cd41813512fc12d7b9.

Hevner, March, Park and Ram (2004) 'Design Science in Information Systems Research', *MIS Quarterly*, vol. 28, no. 1, p. 75.

Hochstein, L. and Moser, R. (2017) *Ansible: Up and running automating configuration management and deployment the easy way*, Sebastopol California, O'Reilly Media.

Humble, J. and Farley, D. (2010) *Continuous delivery: Reliable software releases through build, test, and deployment automation*, Upper Saddle River NJ, Addison-Wesley.

Hummer, W., Rosenberg, F., Oliveira, F. and Eilam, T. (2013) 'Testing Idempotence for Infrastructure as Code', *Middleware 2013: ACMIFIPUSENIX 14th International Middleware Conference, Beijing, China, December 9-13, 2013, Proceedings.* Berlin Heidelberg, Springer Berlin Heidelberg; Imprint: Springer, pp. 368–388.

Hüttermann, M. (2012) *DevOps for Developers*, Scholars Portal.

Installing - Ansible Lint Documentation (2024) [Online]. Available at https://ansible.readthedocs.io/projects/lint/installing/ (Accessed 30 March 2024).

Installing Ansible — Ansible Community Documentation (2024) [Online]. Available at https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html (Accessed 30 March 2024).

Introduction - Watchtower (2023) [Online]. Available at https://containrrr.dev/watchtower/introduction/ (Accessed 30 March 2024).

J, P., Pawar, A. R. and V, N. (2017) 'A REVIEW OF EXISTING CLOUD AUTOMATION TOOLS', *Asian Journal of Pharmaceutical and Clinical Research*, vol. 10, no. 13, p. 471.

J. Esch (2015) 'Software-Defined Networking: A Comprehensive Survey', *Proceedings of the IEEE* [Online]. Available at https://www.semanticscholar.org/paper/Software-Defined-Networking%3A-A-Comprehensive-Survey-Esch/0440b8ca90d8e5e0a2b38e825a1d9850214f9749.

Jeanne Gu (2019) *GM EXTRACT, TRANSFORM AND LOAD PLATFORM AS A SERVICE: How General Motors automated provisioning and increased productivity by 11 times,* General Motors.

John, K. and Douglas, R. (2019) 'INFRASTRUCTURE AS CODE–FINAL REPORT' [Online]. Available at https://www.semanticscholar.org/paper/INFRASTRUCTURE-AS-CODE%E2%80%93FINAL-REPORT-Klein-Reynolds/4a809dc2178e4da943c81f3f1d170cd79cdf444c#citing-papers.

Juncal Alonso, Radosław Piliszek and Matija Cankar (2023) 'Embracing IaC Through the DevSecOps Philosophy: Concepts, Challenges, and a Reference Framework', *IEEE Software* [Online]. Available at https://www.semanticscholar.org/paper/Embracing-IaC-Through-the-DevSecOps-Philosophy%3A-and-Alonso-Piliszek/52d05d0be2133e487d62c7001ea58f949190748a.

Kanies, L. (2006) 'Puppet: Next-Generation Configuration Management', *Login: The Usenix Magazine* [Online]. Available at https://www.semanticscholar.org/paper/Puppet%3A-Next-Generation-Configuration-Management-Kanies/74631add77270a219f17699bc339166b6cab5fe9.

Kersken, S. (2019) *IT-Handbuch für Fachinformatiker: Der Ausbildungsbegleiter*, 9th edn, Bonn, Rheinwerk Verlag; Rheinwerk Computing.

Kreutz, D., Ramos, F. M. V., Verissimo, P., Rothenberg, C. E., Azodolmolky, S. and Uhlig, S. (2014) *Software-Defined Networking: A Comprehensive Survey* [Online]. Available at http://arxiv.org/pdf/1406.0440.pdf.

Kropp, M., Meier, A. and Biddle, R. (2016) 'Agile Practices, Collaboration and Experience', in Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S. and Mikkonen, T. (eds) *Product-focused software process improvement,* New York NY, Springer Berlin Heidelberg, pp. 416–431.

Kumara, I., Garriga, M., Romeu, A. U., Di Nucci, D., Palomba, F., Tamburri, D. A. and van den Heuvel, W.-J. (2021) 'The do's and don'ts of infrastructure code: A systematic gray literature review', *Information and Software Technology*, vol. 137, p. 106593 [Online]. DOI: 10.1016/j.infsof.2021.106593.

Labouardy, M. (2021) *Pipeline as code: Continuous delivery with Jenkins, Kubernetes, and Terraform*, Shelter Island NY, Manning Publications Co.

Limoncelli, T. s. A. (2018) 'GitOps: A Path to More Self-service IT', *Queue* [Online]. Available at https://www.semanticscholar.org/paper/GitOps%3A-A-Path-to-More-Self-service-IT-Limoncelli/38613b41d8f66c0612457333a9218c24a745c653.

Lin, J., Ravichandiran, R., Bannazadeh, H. and Leon-Garcia, A. (2015 - 5/15/2015) 'Monitoring and measurement in software-defined infrastructure', *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM).* Ottawa, ON, Canada, 5/11/2015 - 5/15/2015, IEEE, pp. 742–745.

louislam/uptime-kuma (2024) [Online]. Available at https://github.com/louislam/uptime-kuma (Accessed 31 March 2024).

Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M. and Lassenius, C. (2019) 'DevOps in practice: A multiple case study of five companies', *Information and Software Technology*, vol. 114, pp. 217–230.

Lwakatare, L. E., Kuvaja, P. and Oivo, M. (2016) 'Relationship of DevOps to Agile, Lean and Continuous Deployment', in Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S. and Mikkonen, T. (eds) *Product-focused software process improvement,* New York NY, Springer Berlin Heidelberg, pp. 399–415.

M. Akbar, K. Smolander, Sajjad Mahmood and Ahmed Alsanad (2022) 'Toward successful DevSecOps in software development organizations: A decision-making framework', *Information and Software Technology* [Online]. Available at https://www.semanticscholar.org/paper/Toward-successful-DevSecOps-in-software-development-Akbar-Smolander/7a051c8564ed414010edd91fc829444128e22105.

MarketsandMarkets (2024) *Infrastructure as Code Market Share | Forecast & Statistics | Growth Analysis & Opportunities [2030]* [Online]. Available at https://

www.marketsandmarkets.com/Market-Reports/infrastructure-as-code-market-115458264.html (Accessed 2 March 2024).

Morris, K. (2021) *INFRASTRUCTURE AS CODE: Dynamic systems for the cloud age* [Online], [S.l.], O'Reilly Media. Available at https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=2700918.

Peffers, K., Tuunanen, T., Rothenberger, M. A. and Chatterjee, S. (2007) 'A Design Science Research Methodology for Information Systems Research', *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77.

Prometheus (2024) *Overview | Prometheus* [Online]. Available at https://prometheus.io/docs/introduction/overview/ (Accessed 31 March 2024).

Rahman, A., Farhana, E. and Williams, L. (2020) 'The ‘as code’ activities: development anti-patterns for infrastructure as code', *Empirical Software Engineering*, vol. 25, no. 5, pp. 3430–3467.

Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019) 'A systematic mapping study of infrastructure as code research', *Information and Software Technology*, vol. 108, pp. 65–77.

Rahman, A., Parnin, C. and Williams, L. (2019) 'The Seven Sins: Security Smells in Infrastructure as Code Scripts', *International Conference on Software Engineering*, vol. 41, pp. 164–175.

Rahman, A., Rahman, M. R., Parnin, C. and Williams, L. (2021) 'Security Smells in Ansible and Chef Scripts', *ACM Transactions on Software Engineering and Methodology*, vol. 30, no. 1, pp. 1–31.

Rahman, A. and Sharma, T. (uuuu-uuuu) 'Lessons from Research to Practice on Writing Better Quality Puppet Scripts', *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Honolulu, HI, USA, 3/15/2022 - 3/18/2022, IEEE, pp. 63–67.

Rahman, A. A. U., Helms, E., Williams, L. and Parnin, C. (2015) 'Synthesizing Continuous Deployment Practices Used in Software Development', *2015 Agile Conference (AGILE)*. National Harbor, MD, USA, 8/3/2015 - 8/7/2015, IEEE / Institute of Electrical and Electronics Engineers Incorporated, pp. 1–10.

Rajapakse, R. N., Zahedi, M., Babar, M. A. and Shen, H. (2022) 'Challenges and solutions when adopting DevSecOps: A systematic review', *Information and Software Technology*, vol. 141, p. 106700.

Rich Mogull, James Arlen, Francoise Gilbert, Adrian Lane, David Mortman, Gunnar Peterson and Mike Rothman (2017) *Security Guidance for Critical Areas of Focus in Cloud Computing v4.0,* Cloud Security Alliance.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J. and Männistö, T. (2016) 'DevOps Adoption Benefits and Challenges in Practice: A Case Study', in Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S. and Mikkonen, T. (eds) *Product-focused software process improvement,* New York NY, Springer Berlin Heidelberg, pp. 590–597.

Robbins, J. (2009) *Announcing Chef* [Online], Chef. Available at https://www.chef.io/blog/announcing-chef (Accessed 12 March 2024).

Ronan Keenan, Nigel Kersten and Caitlyn O'Connell (2023) *State of Devops Report 2023,* Puppet by Perforce.

Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P. K. Gupta, Somya Garg and Saransh Ahlawat (2021) 'On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps', *International Conference on Artificial Intelligence and Knowledge Engineering* [Online]. Available at https://www.semanticscholar.org/paper/On-Continuous-Integration-Continuous-Delivery-for-Garg-Pundir/fe3ee5a96d8d6ace5781859d8460cb3db139a568.

Saumya Gupta, Madhulika Bhatia, Meenakshi Memoria and Preeti Manani (2022) 'Prevalence of GitOps, DevOps in Fast CI/CD Cycles', *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)* [Online]. Available at https://www.semanticscholar.org/paper/Prevalence-of-GitOps%2C-DevOps-in-Fast-CI-CD-Cycles-Gupta-Bhatia/0617b9c86c30b46275292d6b09886aed8e236f1f.

Schwarz, J., Steffens, A. and Lichter, H. (2018) 'Code Smells in Infrastructure as Code', *Quality of Information and Communications Technology* [Online]. Available at https://www.semanticscholar.org/paper/Code-Smells-in-Infrastructure-as-Code-Schwarz-Steffens/c4fd608f943a639e35eb13ef7beee1a2f0c30bba.

Spinellis, D. (2012) 'Don't Install Software by Hand', *IEEE Software*, vol. 29, no. 4, pp. 86–87.

Stack Overflow (2024) *Stack Overflow Developer Survey 2023* [Online]. Available at https://survey.stackoverflow.co/2023/#methodology (Accessed 3 April 2024).

Stevens, L. (2017) 'Amazon Finds the Cause of Its AWS Outage: A Typo', *The Wall Street Journal*, 3 February [Online]. Available at https://www.wsj.com/articles/amazon-finds-the-cause-of-its-aws-outage-a-typo-1488490506 (Accessed 1 March 2024).

Stray, V., Fægri, T. E. and Moe, N. B. (2016) 'Exploring Norms in Agile Software Teams', in Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S. and Mikkonen, T. (eds) *Product-focused software process improvement,* New York NY, Springer Berlin Heidelberg, pp. 458–467.

Sync (2021) *Infrastructure as Code Security Insights: SNYK RESEARCH REPORT*.

Teppan, H., Fla, L. H. and Jaatun, M. G. (uuuu-uuuu) 'A Survey on Infrastructure-as-Code Solutions for Cloud Development', *2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom).* Bangkok, Thailand, 12/13/2022 - 12/16/2022, IEEE, pp. 60–65.

Tomas, N., Li, J. and Huang, H. (2019) 'An Empirical Study on Culture, Automation, Measurement, and Sharing of DevSecOps', *International Conference on Cyber Security And Protection Of Digital Services* [Online]. Available at https://www.semanticscholar.org/paper/An-Empirical-Study-on-Culture%2C-Automation%2C-and-of-Tomas-Li/e152aaff5ad9fcb8095c644e015df9b198e4b7f4.

Traefik.io (2024a) *Routing & Load Balancing Overview |Traefik Docs - Traefik* [Online]. Available at https://doc.traefik.io/traefik/routing/overview/ (Accessed 30 March 2024).

Traefik.io (2024b) *Traefik Proxy Documentation - Traefik* [Online]. Available at https://doc.traefik.io/traefik/ (Accessed 30 March 2024).

What is a Homelab and what does it do? : r/homelab (2024) [Online]. Available at https://www.reddit.com/r/homelab/comments/8q1sz7/what_is_a_homelab_and_what_does_it_do/ (Accessed 20 March 2024).

Wieringa, R. J. (2014) *Design science methodology for information systems and software engineering*, New York NY, Springer Berlin Heidelberg.

Williams, L. and Vouk, M. (2009) 'Agile Software Development', in Wah, B. W. (ed) *Wiley encyclopedia of computer science and engineering,* Hoboken N.J., John Wiley.

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K. and Soldani, J. (2020) 'The essential deployment metamodel: a systematic review of deployment automation technologies', *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, 1-2, pp. 63–75.

Yuen, B. (2021) *GitOps and Kubernetes: Continuous development with Argo CD, Jenkins X and Flux*, Shelter Island, Manning Publications.

V. Appendix

Table 3: Comprehensive Inventory of IT Department Functions Aligned with Case Study Implementation.

Category	Function	Description	Popular Examples	Use in SME	Deployed in Case Study
Communication	Groupware and Email	Collaboration through shared calendar and email services.	Microsoft Exchange, Postfix, Zimbra	Extensive	Exchange Online, Microsoft 365
Communication	Team Collaboration Messaging	Enables messaging and collaboration among team members.	Slack, Microsoft Teams, Google Hangouts	High	Microsoft 365
Enterprise Handling	Accounting and Financial Management	Manages financial transactions and reporting.	DATEV, Lexware, SAP Business One	Extensive	IHateMoney
Enterprise Handling	BI (Business Intelligence)	Analyzes data for business decisions.	Power BI, Tableau, SAP BusinessObjects	Moderate	/
Enterprise Handling	CMS (Content Management System)	Enables managing web content without technical knowledge.	WordPress, TYPO3, Drupal	High	Ghost
Enterprise Handling	CRM (Customer Relationship Management)	Manages interactions with customers.	Salesforce, SAP CRM, Microsoft Dynamics 365	Common	/
Enterprise Handling	ERP (Enterprise Resource Planning)	Integrates data and processes of an organization.	SAP ERP, Oracle ERP, Microsoft Dynamics AX	Common	/
Enterprise Handling	HRM (Human Resources Management)	Manages organization's human resources processes.	P&I Loga, Haufe umantis, SAP SuccessFactors	Common	/
Enterprise Handling	Project Management Tool	Organizes and manages project resources.	Trello, Asana, Microsoft Project	Moderate	/
Enterprise Handling	SCM (Supply Chain Management)	Manages the flow of goods and information.	SAP SCM, Oracle SCM, Infor CloudSuite SCM	Moderate	/
Hardware	Endpoint	Primary user-interface devices for network interaction.	Desktop PCs, Laptops, iPhones, Android phones	Extensive	Multiple Devices
Hardware	Server Computer	Essential computing components for network services.	Dell PowerEdge, HPE ProLiant	Common	Custom Build Server
Hardware	Switching	Connects and manages devices on a network.	Cisco Switches, NETGEAR Switches	Extensive	Cisco

Hardware	Telephone	Enables voice communication.	Avaya IP Office, Siemens OpenStage	Extensive	FRITZ!Box
Hardware	UPS (Uninterruptible Power Supply)	Provides emergency power during power source failures.	APC, CyberPower	Common	BlueWalker
Hardware	Wi-Fi Access Point	Provides wireless networking connectivity.	Cisco, Aruba, Netgear	High	Ubiquiti
IT Service	Code Repository	Manages and stores software code.	GitHub, GitLab, Bitbucket	Moderate	GitHub, Code-Server
IT Service	Endpoint and Server Management	Managing updates and settings.	Microsoft WSUS, SCCM, GPO	High	/
IT Service	Hypervisor	Creates and manages virtual machines.	VMware vSphere, Microsoft Hyper-V	Common	Proxmox Virtual Environment
IT Service	IT Ticket Management	Tracks the resolution of IT and customer support issues.	Jira Service Desk, OTRS, Freshservice	Common	/
IT Service	Print Server	Manages printers and print jobs in a network.	HP Print Servers, CUPS	Common	/
IT Service	Remote Desktop	Enables remote usage of a computers.	Citrix, TeamViewer	Moderate	/
IT Service	System Monitoring and Logging	Observes network and component health.	Nagios, Zabbix, SolarWinds	Extensive	Prometheus, Uptime Kuma
Knowledge and Storage	Data Backup and Recovery	Protects against data loss by storing copies of data.	Veeam, Acronis, Nakivo	Extensive	Proxmox Backup Server, Rsync
Knowledge and Storage	DMS (Document Management System)	Organizes and retrieves documents within an organization.	SharePoint, DocuWare	High	Paperless-ngx, Calibre
Knowledge and Storage	LMS (Learning Management System)	Manages educational courses and training programs.	Moodle, Blackboard, Canvas	Minimal	/
Knowledge and Storage	Network Storage	Provides centralized data storage on a network.	NetApp, Synology, QNAP	Extensive	OpenMediaVault, Filebrowser
Networking	DHCP and DNS	Manages and translates IP addresses.	Infoblox, BIND, Microsoft DHCP	Extensive	AdGuard Home
Networking	SIP Gateway	Connects voice communications with phone networks.	Avaya, Snom	High	/

Networking	VPN	Connects remote users and offices.	Cisco Anyconnect, FortiClient	High	WireGuard
Networking	Web Server and Reverse Proxy	Stores, processes, and delivers web pages.	Apache, Nginx, Microsoft IIS	Extensive	Traefik, NodeJS, Apache
Security	Antivirus Software	Protects against malware and cyber threats.	Norton, Trend Micro, Kaspersky	Common	/
Security	Domain Management and IAM	Manages network user authentication and access.	Microsoft Active Directory	Extensive	Microsoft Entra ID, Authelia
Security	NGFW (Next-Generation Firewall)	Protects network traffic based on security rules.	Sophos, Fortinet	Extensive	OPNsense
Security	Password Management	Secures and manages enterprise passwords.	Password Safe, Pleasant Password Server, Keepass	High	Vaultwarden

Figure 25

Table 4: Overview of All Case Study Hosts and Their Respective Services with Determined Automation Levels.

Host	Application	Category	Description	Automation Level	Status
Openmediavault	Adguard Exporter	Monitoring	Collects AdGuard's metrics	Fully	Done
1blu SX VPS	Adguard Home*	Networking	DNS Filtering and Rewrites	Fully	Pending
DNS LXC1	Adguard Home*	Networking	DNS Filtering and Rewrites	Fully	Pending
DNS LXC2	Adguard Home*	Networking	DNS Filtering and Rewrites	Fully	Pending
Openmediavault	AdguardSync	Utility	Syncs AdGuard settings across instances	Fully	Done
Azure B1ls (VM)	Alertmanager	Monitoring	Handles alerts from Prometheus	Fully	Done
Openmediavault	Alertmanager	Monitoring	Handles alerts from Prometheus	Fully	Done
Azure B1ls (VM)	Apache	Web Application	Personal Website	Partially	Pending
Azure B1ls (VM)	Authelia	Authentication	Single-sign-on authentication server	Fully	Pending
Azure B1ls (VM)	Autoscan	Media-Utility	Automated media scan tool	Fully	Pending
Google Storage	Backup Backend*	Storage	Cloud storage as backup backend	Partially	Excluded
Proxmox Backup Server	Backup Solution*	Backup	VM, Container and host backup tool	Partially	Pending
Openmediavault	Bazarr Exporter	Monitoring	Collects Bazarr's metrics	Fully	Done
Azure B1ls (VM)	BetterGPT	Utility	Alternative ChatGPT Frontend	Partially	Pending
1blu SX VPS	cAdvisor	Monitoring	Collects container metrics	Fully	Done
Azure B1ls (VM)	cAdvisor	Monitoring	Collects container metrics	Fully	Done
Openmediavault	cAdvisor	Monitoring	Collects container metrics	Fully	Done
Ubuntu Server	cAdvisor	Monitoring	Collects container metrics	Fully	Done
Azure B1ls (VM)	Calibre	Media	E-book management software	Fully	Pending
Azure B1ls (VM)	Calibre-Web	Media	Web-based e-book reader	Fully	Pending
1blu SX VPS	Certdumper	Utility	Dumps SSL certificates	Fully	Done

Azure B1ls (VM)	Certdumper	Utility	Dumps SSL certificates	Fully	Done
Openmediavault	Certdumper	Admin	Dumps SSL certificates	Fully	Done
Openmediavault	Changedetection	Utility	Tracks website changes	Partially	Done
1blu SX VPS	Cloudplow	Media-Utility	Media file processing utility	Fully	Pending
Openmediavault	Code-Server	DevOps	Web-based VS Code instance	Partially	Done
Openmediavault	Czkawka	Utility	Finds duplicate files	Fully	Done
OPNsense	DHCP*	Networking	Assigns IP addresses in network	Fully	Pending
1blu SX VPS	Docker-Socket-Proxy	Monitoring	Proxies Docker API	Fully	Done
Azure B1ls (VM)	Docker-Socket-Proxy	Monitoring	Proxies Docker API	Fully	Done
Openmediavault	Docker-Socket-Proxy	Monitoring	Proxies Docker API	Fully	Done
Ubuntu Server	Docker-Socket-Proxy	Monitoring	Proxies Docker API	Fully	Done
Openmediavault	Dockge	Admin	Docker Compose Manager	Partially	Done
Azure B1ls (VM)	Filebrowser	Media	Web-based file manager	Partially	Pending
Azure B1ls (VM)	Ghost	CMS	Modern blogging platform	Partially	Pending
Openmediavault	Grafana	Monitoring	Visualizes metrics	Partially	Done
Azure B1ls (VM)	Headscale	Networking	Tailscale-compatible control server	Partially	Pending
Azure B1ls (VM)	Homer	Web Application	Customizable start page	Fully	Pending
Azure B1ls (VM)	IHateMoney	Finance	Personal finance tool	Partially	Pending
1blu SX VPS	Librespeed	Monitoring	Measures network performance	Fully	Done
Azure B1ls (VM)	Librespeed	Monitoring	Measures network performance	Fully	Done
Openmediavault	Librespeed	Monitoring	Measures network performance	Fully	Done
Openmediavault	Librespeed Exporter	Monitoring	Performs network speedtests	Fully	Done
Azure B1ls (VM)	Linkstack	Web Application	Link sharing platform	Fully	Pending
Openmediavault	NAS Server*	Storage	Provides networked file storage	Partially	Done

1blu SX VPS	Node Exporter	Monitoring	Collects OS metrics	Fully	Done
Azure B1ls (VM)	Node Exporter	Monitoring	Collects OS metrics	Fully	Done
Homeassistant	Node Exporter	Monitoring	Collects OS metrics	Manual	Done
Openmediavault	Node Exporter	Monitoring	Collects OS metrics	Fully	Done
OPNsense	Node Exporter	Monitoring	Collects OS metrics	Fully	Pending
Ubuntu Server	Node Exporter	Monitoring	Collects OS metrics	Fully	Done
Azure B1 (Web App)	NodeJS*	Web Application	Custom Website for a non-profit	Partially	Pending
Azure B1ls (VM)	NodeJS*	Web Application	Demo Website for a Uni project	Partially	Pending
OPNsense	NTP*	Networking	Synchronizes network time	Fully	Pending
Azure B1ls (VM)	Overseerr	Media-Utility	Request management for Plex	Partially	Pending
Openmediavault	Paperless-ngx	DMS	Manages digital documents	Partially	Done
Ubuntu Server	Photoprism	Media	Personal photo management	Partially	Pending
Azure B1ls (VM)	phpMyAdmin	Admin	Manages MySQL databases	Fully	Pending
Azure B1ls (VM)	Plausible	Web Analytics	Privacy-respecting analytics	Partially	Pending
Openmediavault	Plausible Exporter	Monitoring	Collects Plausible's metrics	Fully	Done
1blu SX VPS	Plex	Media	Media server software	Partially	Pending
Openmediavault	Plex Exporter	Monitoring	Collects Radarr's metrics	Fully	Done
Openmediavault	Portainer	Admin	Manages Docker environments	Partially	Done
1blu SX VPS	Portainer Agent	Admin	Agent for Portainer	Fully	Pending
Azure B1ls (VM)	Portainer Agent	Admin	Agent for Portainer	Fully	Pending
Ubuntu Server	Portainer Agent	Admin	Agent for Portainer	Fully	Pending
Azure B1ls (VM)	Prometheus	Monitoring	Analyzes and stores metrics	Fully	Pending
Openmediavault	Prometheus	Monitoring	Analyzes and stores metrics	Fully	Done
Azure B1ls (VM)	Prowlarr	Media-Utility	Automates media searches	Partially	Pending

Openmediavault	Prowlarr Exporter	Monitoring	Collects Prowlarr's metrics	Fully	Done
Openmediavault	Proxmox Exporter	Monitoring	Collects Proxmox's metrics	Fully	Done
Openmediavault	pyLoad	Media	Lightweight download manager	Partially	Done
Openmediavault	qBittorrent Exporter	Monitoring	Collects qBittorrent's metrics	Fully	Done
Azure B1ls (VM)	Radarr	Media	Automates movie downloads	Partially	Pending
Openmediavault	Radarr Exporter	Monitoring	Collects Radarr's metrics	Fully	Done
1blu SX VPS	Rclone	Storage-Utility	Syncs files to cloud storage	Fully	Pending
Azure B1ls (VM)	Rclone	Storage-Utility	Syncs files to cloud storage	Fully	Pending
Azure B1ls (VM)	Recyclarr	Media-Utility	Automates media profile creation	Fully	Pending
OPNsense	Routing*	Networking	Firewall and routing	Fully	Done
1blu SX VPS	Sabnzbd	Media	Automated media downloader	Partially	Pending
Openmediavault	Sabnzbd Exporter	Monitoring	Collects Sabnzbd's metrics	Fully	Done
Azure B1ls (VM)	Searxng	Web Application	Privacy-respecting search engine	Fully	Pending
Homeassistant	Smart Home Server*	Smart Home	Controls smart home devices	Partially	Pending
1blu SX VPS	Socks5-Proxy	Monitoring	Proxies web requests	Fully	Done
Azure B1ls (VM)	Socks5-Proxy	Monitoring	Proxies web requests	Fully	Done
Azure B1ls (VM)	Sonarr	Media	Automates TV show downloads	Partially	Pending
Openmediavault	Sonarr Exporter	Monitoring	Collects Sonarr's metrics	Fully	Done
Openmediavault	Tandoor Recipes	ERP	Recipe management solution	Fully	Done
Azure B1ls (VM)	Tautulli	Media-Utility	Monitors Plex media server	Fully	Pending
1blu SX VPS	Traefik	Networking	Reverse proxy and load balancer	Fully	Done
Azure B1ls (VM)	Traefik	Networking	Reverse proxy and load balancer	Fully	Done
Openmediavault	Traefik	Networking	Reverse proxy and load balancer	Fully	Done
Ubuntu Server	Traefik	Networking	Reverse proxy and load balancer	Fully	Done

Openmediavault	Unifi Controller	Networking	Manages Ubiquiti UniFi devices	Partially	Done
Openmediavault	Unifi Exporter	Monitoring	Collects UniFi's metrics	Fully	Done
Azure B1ls (VM)	Uptimekuma	Monitoring	Monitors uptime of endpoints	Fully	Pending
Openmediavault	Uptimekuma	Monitoring	Monitors uptime of endpoints	Fully	Done
Openmediavault	Valetudo	Smart Home	Generates robovacuums map	Fully	Done
Azure B1ls (VM)	Vaultwarden	Password Manager	Self-hosted Bitwarden password server	Fully	Pending
1blu SX VPS	Watchtower	Utility	Updates running Docker containers	Fully	Done
Azure B1ls (VM)	Watchtower	Utility	Updates running Docker containers	Fully	Done
Openmediavault	Watchtower	Utility	Updates running Docker containers	Fully	Done
Ubuntu Server	Watchtower	Utility	Updates running Docker containers	Fully	Done
Azure B1ls (VM)	WireGuard VPN*	Networking	Secure VPN server	Fully	Pending
OPNsense	WireGuard VPN*	Networking	VPN server	Fully	Pending
Azure B1ls (VM)	Wizarr	Media-Utility	User Management for Plex	Partially	Pending
Azure B1ls (VM)	xBackBone	Media	Shares screen recordings	Fully	Pending
1blu SX VPS	Xteve	Media	IPTV provider proxy	Fully	Pending
Azure B1ls (VM)	Your Spotify	Web Application	Personalized Spotify usage dashboard	Fully	Pending
Microsoft SaaS	Microsoft 365*	Groupware	Integrated Microsoft office suite	Manual	Excluded
Microsoft SaaS	Exchange Online*	Groupware	Cloud email server	Manual	Excluded
Microsoft SaaS	Azure Entra ID*	Authentication	Identity and access management service	Manual	Excluded

Open-Sourcing Infrastructure as Code (IaC) Projects

Results

Survey 661976

Number of records in this query:	81
Total records in survey:	81
Percentage of total:	100.00%

Summary for G01Q01

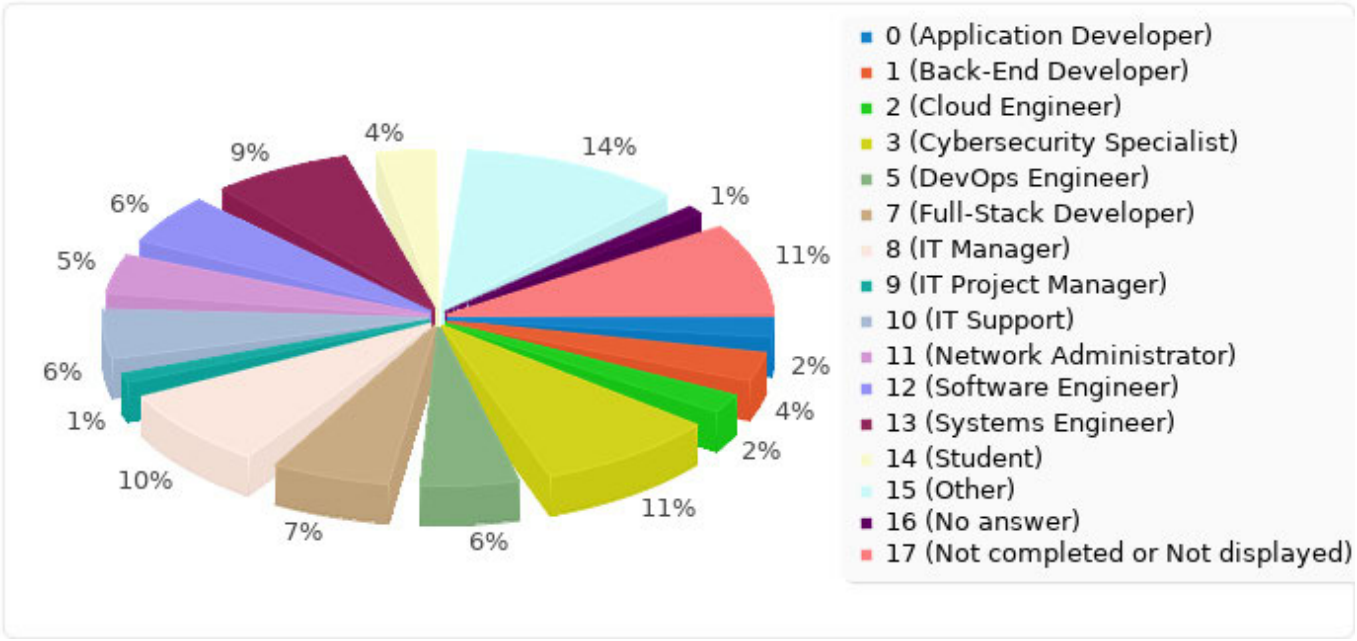
What is your current role in the IT industry?

Answer	Count	Percentage
Application Developer (AO1)	2	2.47%
Back-End Developer (AO2)	3	3.70%
Cloud Engineer (AO3)	2	2.47%
Cybersecurity Specialist (AO4)	9	11.11%
Database Administrator (AO5)	0	0.00%
DevOps Engineer (AO6)	5	6.17%
Front-End Developer (AO7)	0	0.00%
Full-Stack Developer (AO8)	6	7.41%
IT Manager (AO9)	8	9.88%
IT Project Manager (AO10)	1	1.23%
IT Support (AO11)	5	6.17%
Network Administrator (AO12)	4	4.94%
Software Engineer (AO13)	5	6.17%
Systems Engineer (AO14)	7	8.64%
Student (AO15)	3	3.70%
Other	11	13.58%
No answer	1	1.23%
Not completed or Not displayed	9	11.11%

ID	Response
23	Not
32	Data Engineer
33	Data Software Engineer
44	Hobbyist
48	worker who change cartriges
49	None. Just homelab
51	Freshly graduated in IT
57	None
59	Cloud Architect
61	NOC Technician
63	Just Hobby Homelab

Summary for G01Q01

What is your current role in the IT industry?



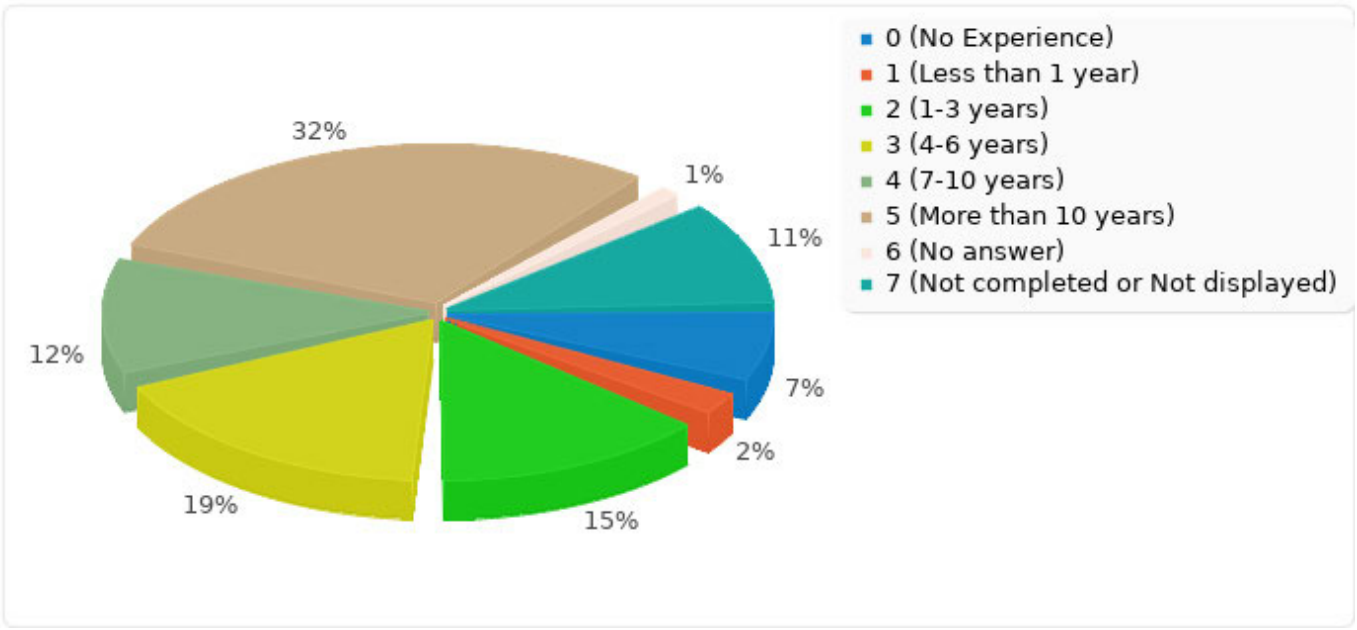
Summary for G01Q02

How many years of experience do you have in the IT industry?

Answer	Count	Percentage
No Experience (AO01)	6	7.41%
Less than 1 year (AO02)	2	2.47%
1-3 years (AO03)	12	14.81%
4-6 years (AO04)	15	18.52%
7-10 years (AO05)	10	12.35%
More than 10 years (AO06)	26	32.10%
No answer	1	1.23%
Not completed or Not displayed	9	11.11%

Summary for G01Q02

How many years of experience do you have in the IT industry?



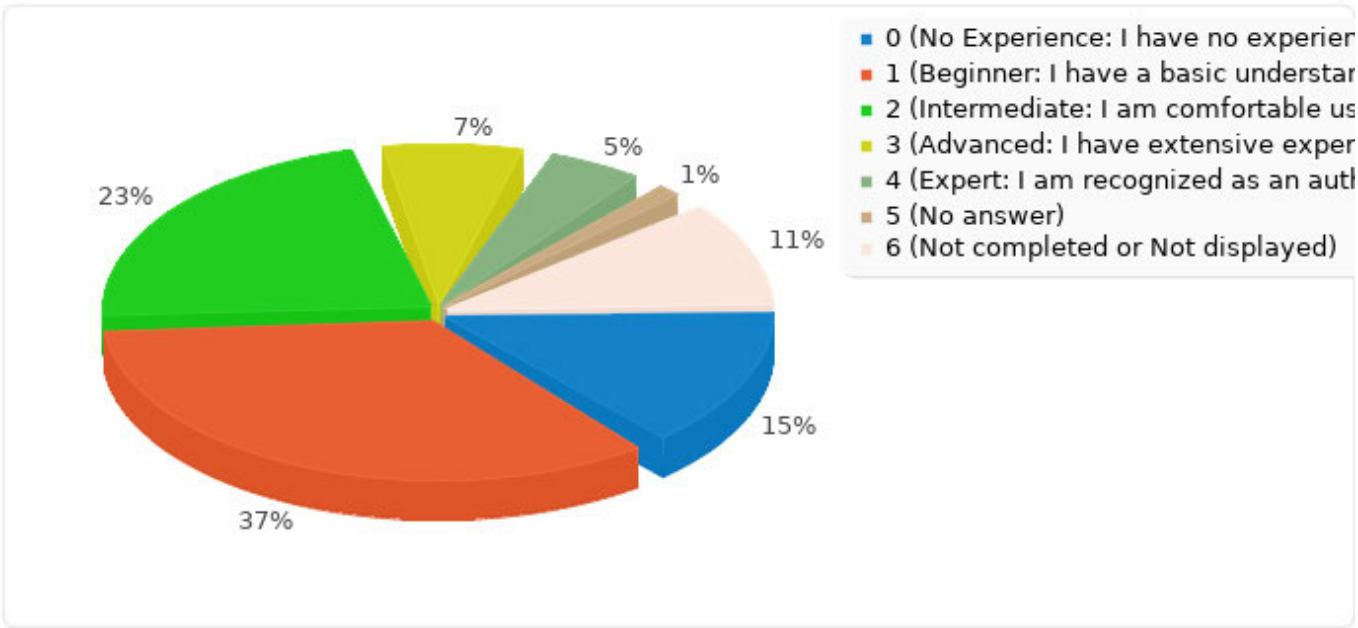
Summary for G01Q03

How would you rate your skill level with Infrastructure as Code tools and practices?

Answer	Count	Percentage
No Experience: I have no experience with IaC practices. (AO01)	12	14.81%
Beginner: I have a basic understanding and limited experience with IaC. (AO02)	30	37.04%
Intermediate: I am comfortable using IaC tools and have completed several projects. (AO03)	19	23.46%
Advanced: I have extensive experience and can design complex IaC solutions. (AO04)	6	7.41%
Expert: I am recognized as an authority on IaC, with deep expertise and leadership in IaC initiatives. (AO05)	4	4.94%
No answer	1	1.23%
Not completed or Not displayed	9	11.11%

Summary for G01Q03

How would you rate your skill level with Infrastructure as Code tools and practices?



Summary for G02Q04

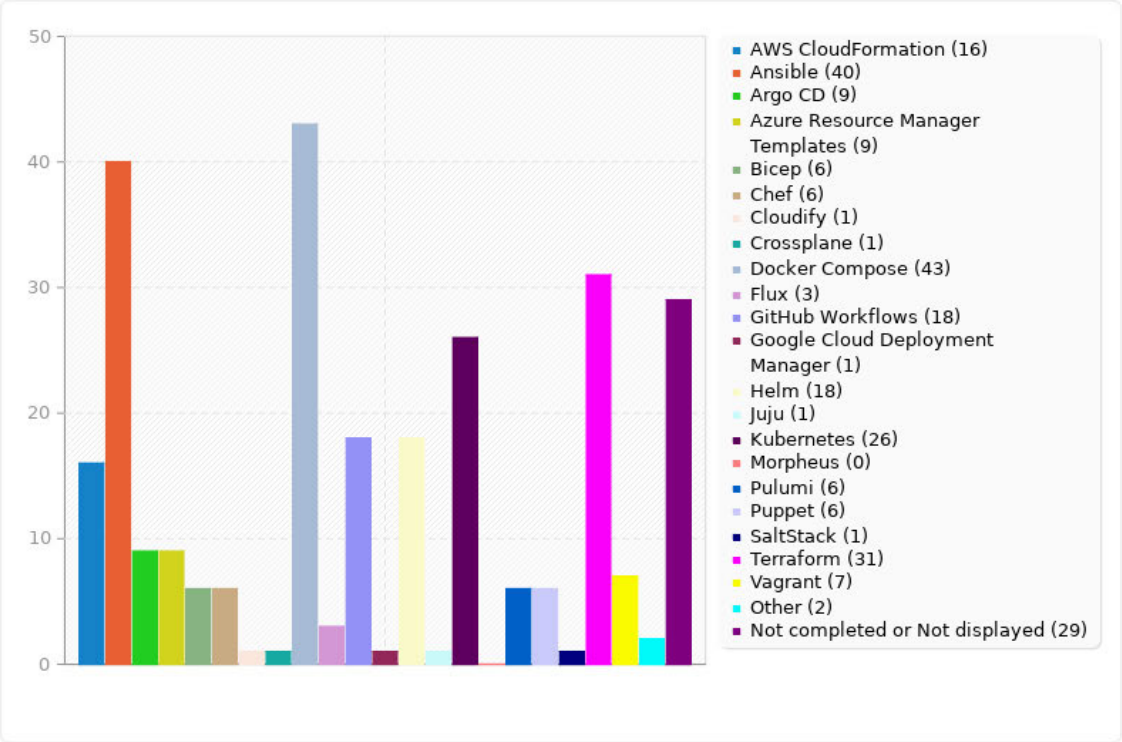
Which IaC tools and technologies have you used?

Answer	Count	Percentage
AWS CloudFormation (SQ001)	16	19.75%
Ansible (SQ002)	40	49.38%
Argo CD (SQ003)	9	11.11%
Azure Resource Manager Templates (SQ004)	9	11.11%
Bicep (SQ005)	6	7.41%
Chef (SQ006)	6	7.41%
Cloudify (SQ007)	1	1.23%
Crossplane (SQ008)	1	1.23%
Docker Compose (SQ009)	43	53.09%
Flux (SQ010)	3	3.70%
GitHub Workflows (SQ011)	18	22.22%
Google Cloud Deployment Manager (SQ012)	1	1.23%
Helm (SQ013)	18	22.22%
Juju (SQ014)	1	1.23%
Kubernetes (SQ015)	26	32.10%
Morpheus (SQ016)	0	0.00%
Pulumi (SQ017)	6	7.41%
Puppet (SQ018)	6	7.41%
SaltStack (SQ019)	1	1.23%
Terraform (SQ020)	31	38.27%
Vagrant (SQ021)	7	8.64%
Other	2	2.47%
Not completed or Not displayed	29	35.80%

ID	Response
25	Packer
33	Custom Python dev

Summary for G02Q04

Which IaC tools and technologies have you used?



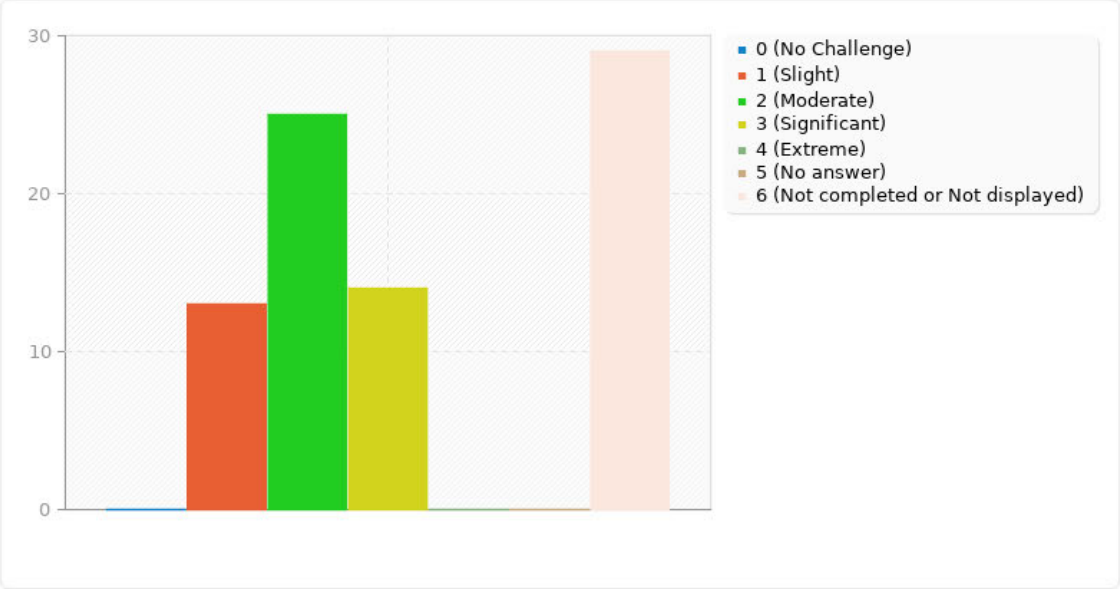
Summary for G02Q05(SQ001)[Complexity of tools and technologies]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	0	0.00%
Slight (AO03)	13	16.05%
Moderate (AO04)	25	30.86%
Significant (AO05)	14	17.28%
Extreme (AO06)	0	0.00%
No answer	0	0.00%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ001)[Complexity of tools and technologies]

Rate the level of challenges you are facing with the following aspects of IaC:



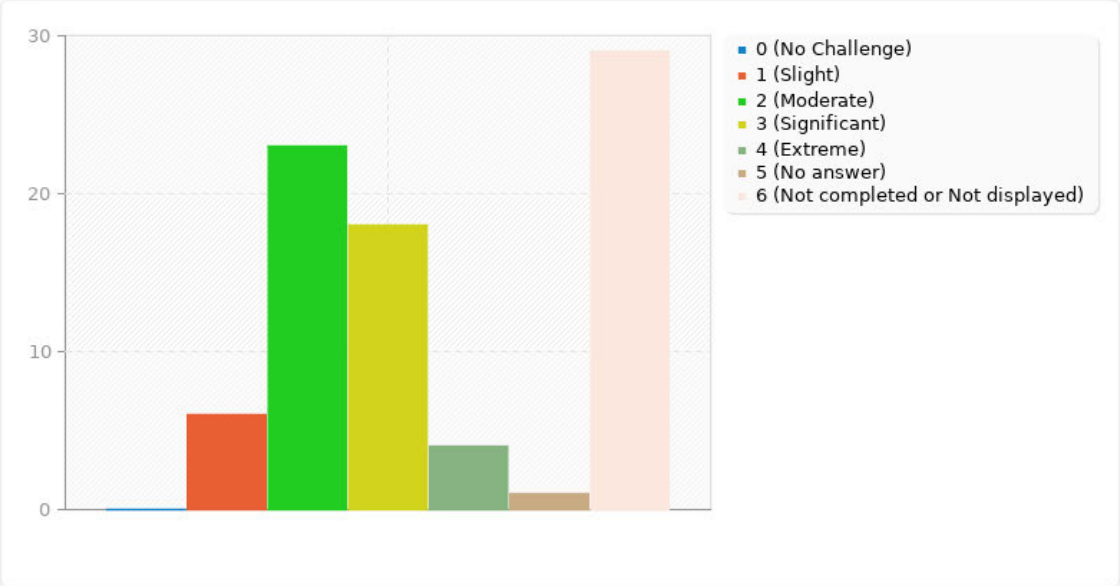
Summary for G02Q05(SQ002)[Ensuring security and compliance within IaC practices]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	0	0.00%
Slight (AO03)	6	7.41%
Moderate (AO04)	23	28.40%
Significant (AO05)	18	22.22%
Extreme (AO06)	4	4.94%
No answer	1	1.23%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ002)[Ensuring security and compliance within IaC practices]

Rate the level of challenges you are facing with the following aspects of IaC:



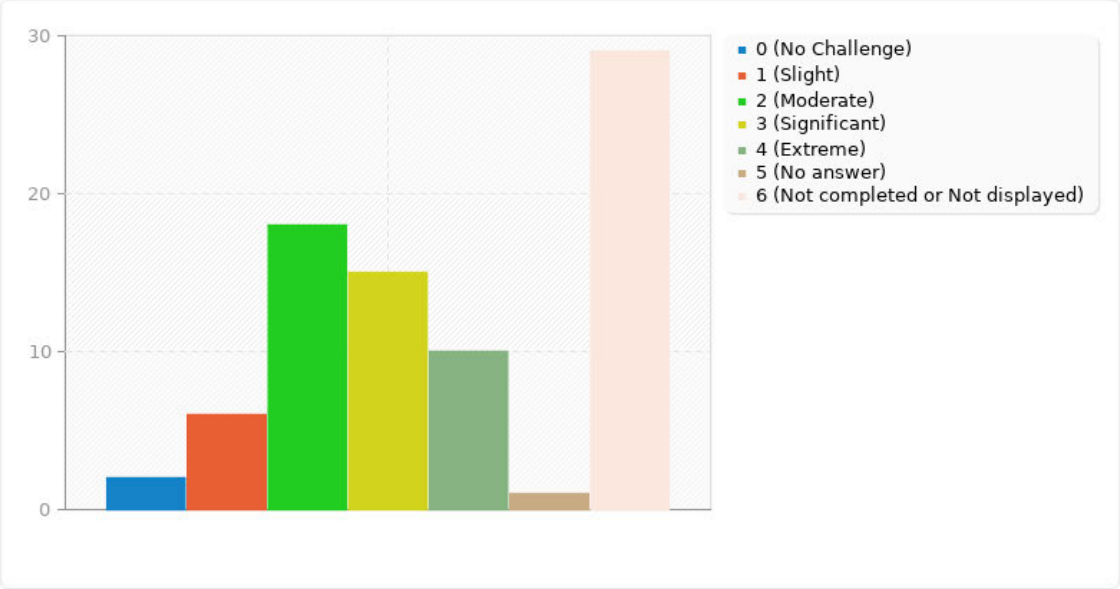
Summary for G02Q05(SQ003)[Integrating IaC with existing infrastructure]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	2	2.47%
Slight (AO03)	6	7.41%
Moderate (AO04)	18	22.22%
Significant (AO05)	15	18.52%
Extreme (AO06)	10	12.35%
No answer	1	1.23%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ003)[Integrating IaC with existing infrastructure]

Rate the level of challenges you are facing with the following aspects of IaC:



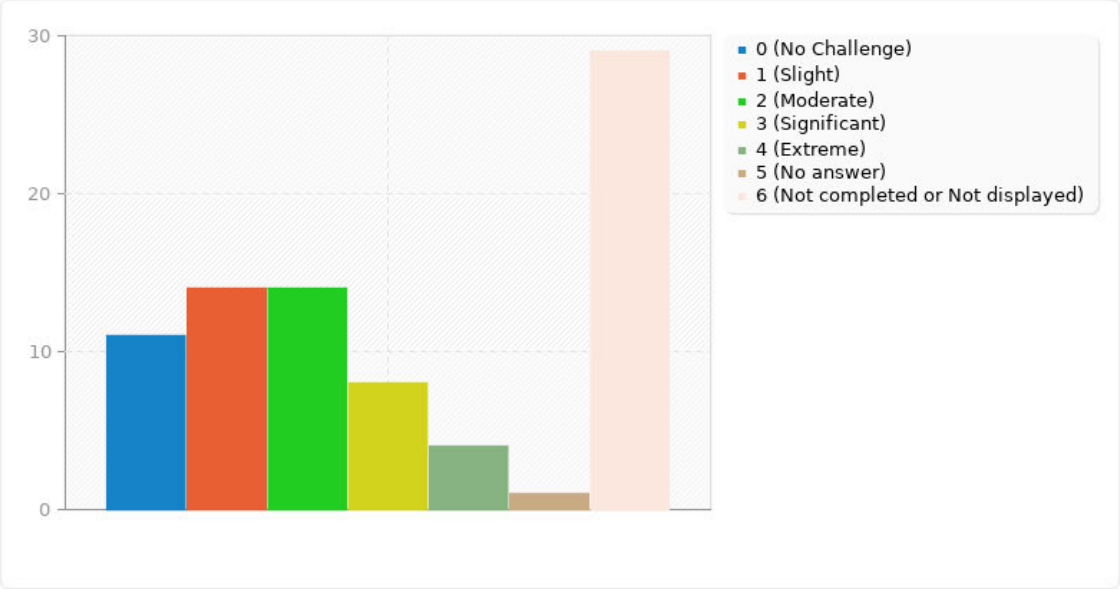
Summary for G02Q05(SQ004)[Keeping IaC code and documentation up to date]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	11	13.58%
Slight (AO03)	14	17.28%
Moderate (AO04)	14	17.28%
Significant (AO05)	8	9.88%
Extreme (AO06)	4	4.94%
No answer	1	1.23%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ004)[Keeping IaC code and documentation up to date]

Rate the level of challenges you are facing with the following aspects of IaC:



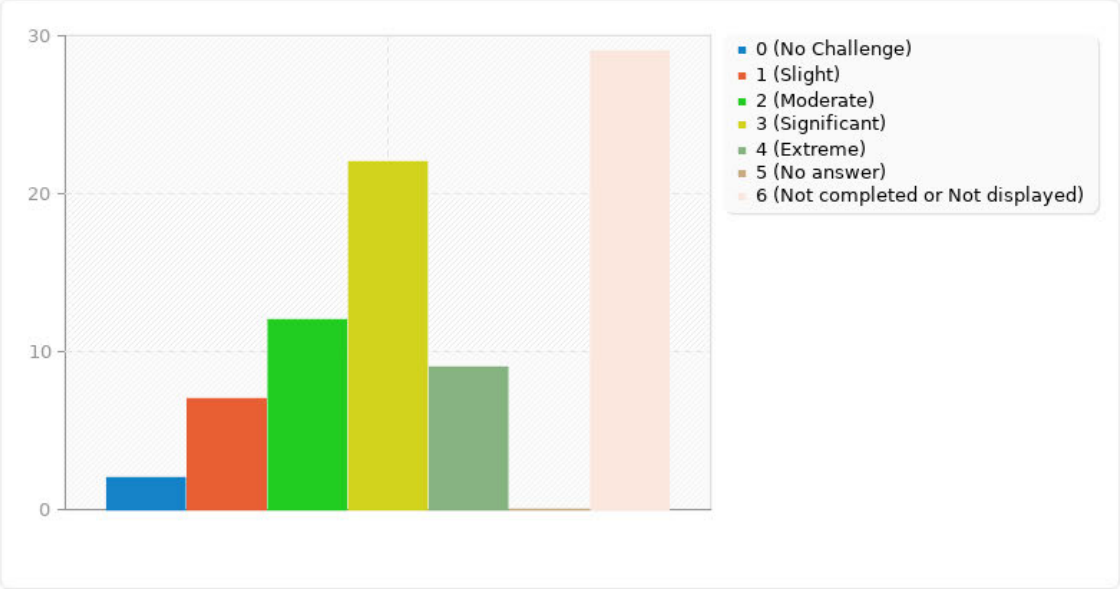
Summary for G02Q05(SQ005)[Lack of expertise]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	2	2.47%
Slight (AO03)	7	8.64%
Moderate (AO04)	12	14.81%
Significant (AO05)	22	27.16%
Extreme (AO06)	9	11.11%
No answer	0	0.00%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ005)[Lack of expertise]

Rate the level of challenges you are facing with the following aspects of IaC:



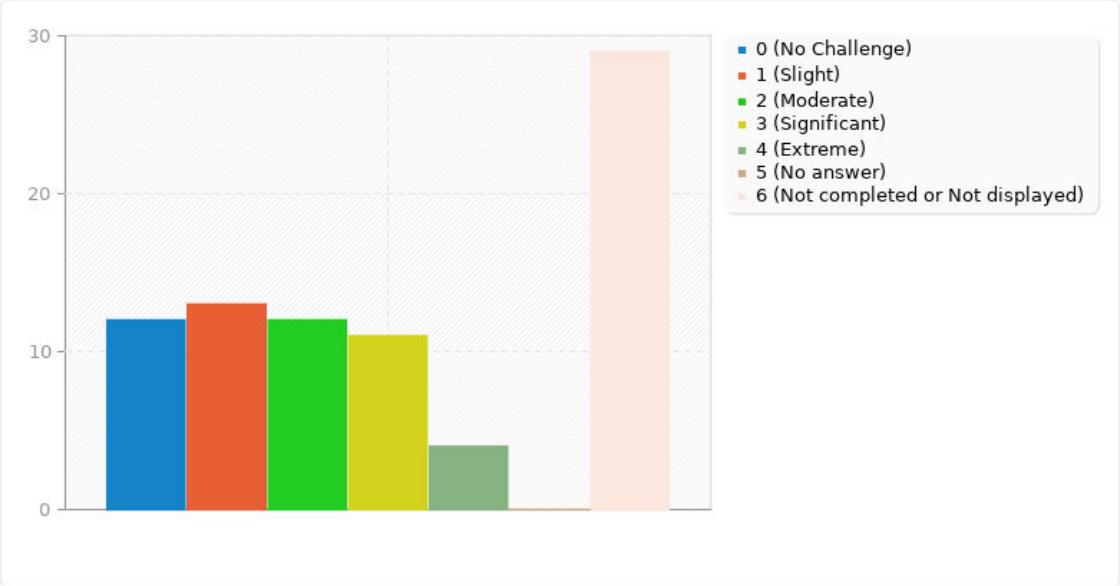
Summary for G02Q05(SQ006)[Limited budget for tools and training]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	12	14.81%
Slight (AO03)	13	16.05%
Moderate (AO04)	12	14.81%
Significant (AO05)	11	13.58%
Extreme (AO06)	4	4.94%
No answer	0	0.00%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ006)[Limited budget for tools and training]

Rate the level of challenges you are facing with the following aspects of IaC:



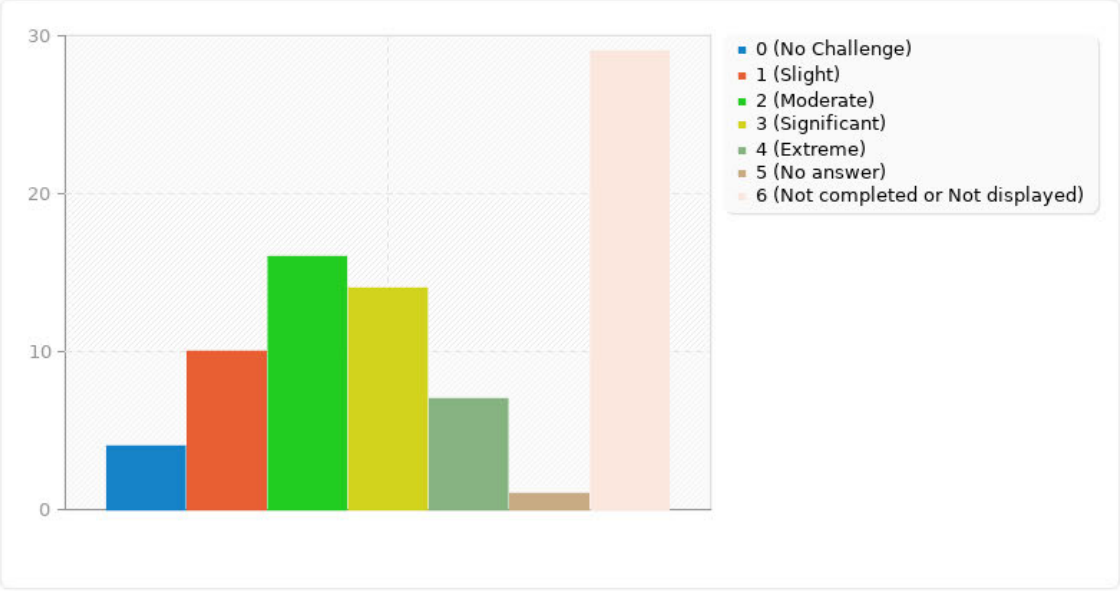
Summary for G02Q05(SQ007)[Monitoring and managing infrastructure drift]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	4	4.94%
Slight (AO03)	10	12.35%
Moderate (AO04)	16	19.75%
Significant (AO05)	14	17.28%
Extreme (AO06)	7	8.64%
No answer	1	1.23%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ007)[Monitoring and managing infrastructure drift]

Rate the level of challenges you are facing with the following aspects of IaC:



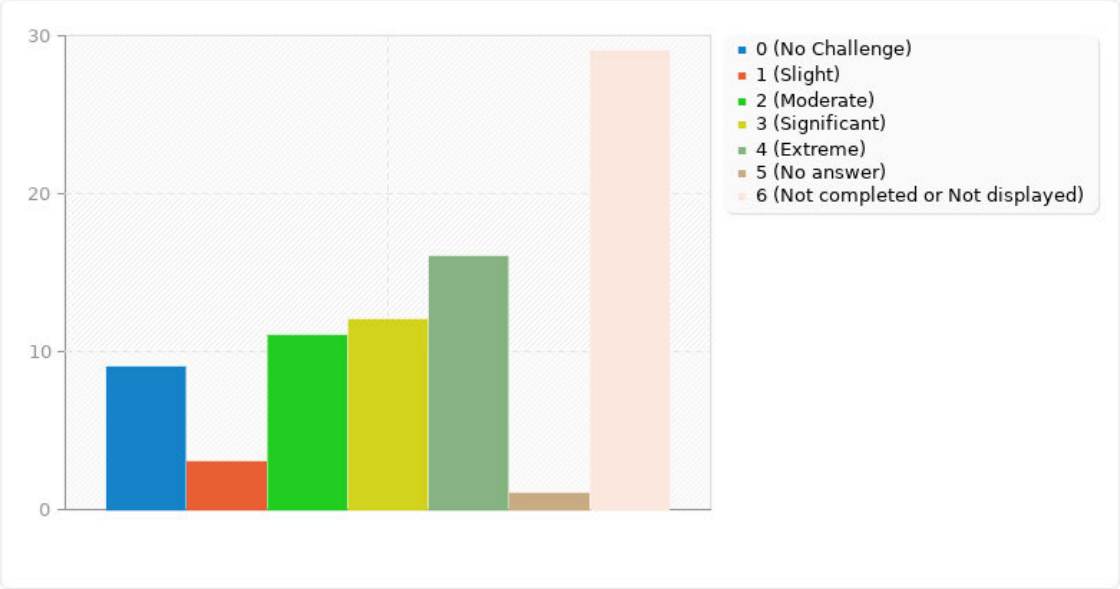
Summary for G02Q05(SQ008)[Resistance to change within the organization]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	9	11.11%
Slight (AO03)	3	3.70%
Moderate (AO04)	11	13.58%
Significant (AO05)	12	14.81%
Extreme (AO06)	16	19.75%
No answer	1	1.23%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ008)[Resistance to change within the organization]

Rate the level of challenges you are facing with the following aspects of IaC:



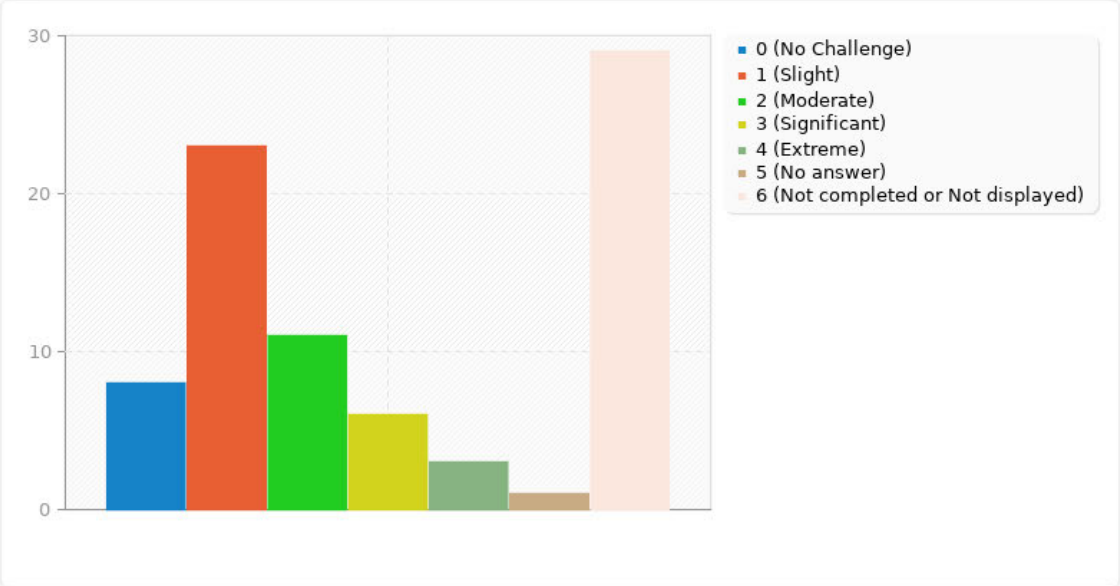
Summary for G02Q05(SQ009)[Scaling IaC for large or growing infrastructures]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	8	9.88%
Slight (AO03)	23	28.40%
Moderate (AO04)	11	13.58%
Significant (AO05)	6	7.41%
Extreme (AO06)	3	3.70%
No answer	1	1.23%
Not completed or Not displayed	29	35.80%

Summary for G02Q05(SQ009)[Scaling IaC for large or growing infrastructures]

Rate the level of challenges you are facing with the following aspects of IaC:



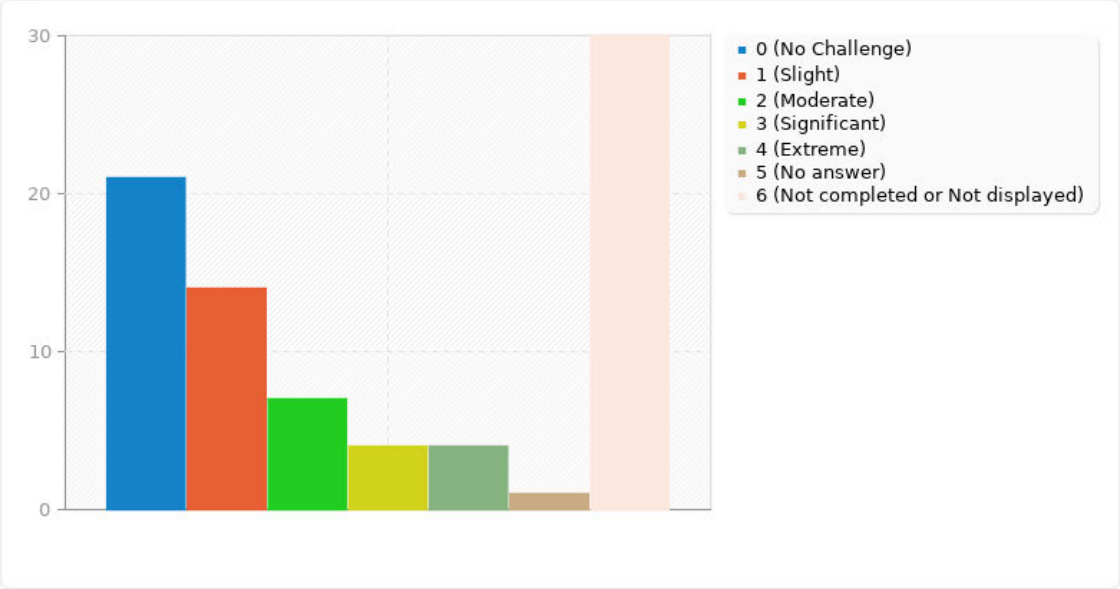
Summary for G02Q05(SQ010)[Version control and collaboration issues]

Rate the level of challenges you are facing with the following aspects of IaC:

Answer	Count	Percentage
No Challenge (AO02)	21	25.93%
Slight (AO03)	14	17.28%
Moderate (AO04)	7	8.64%
Significant (AO05)	4	4.94%
Extreme (AO06)	4	4.94%
No answer	1	1.23%
Not completed or Not displayed	30	37.04%

Summary for G02Q05(SQ010)[Version control and collaboration issues]

Rate the level of challenges you are facing with the following aspects of IaC:



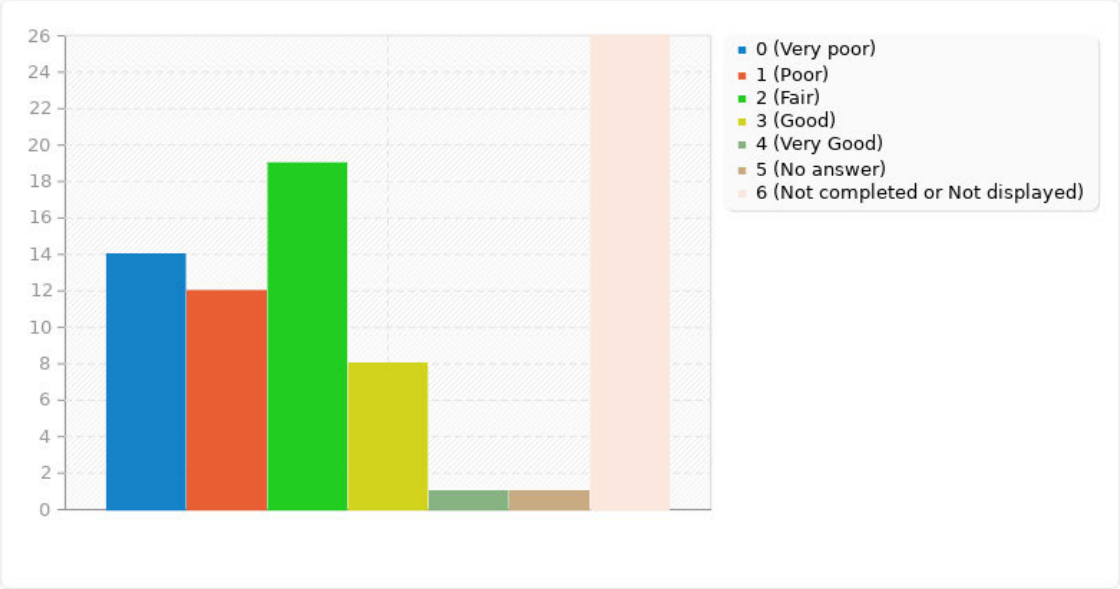
Summary for G03Q06

How would you rate the availability of open-source IaC deployments that comprehensively manage a complete IT environment?

Answer	Count	Percentage
Very poor (AO01)	14	17.28%
Poor (AO02)	12	14.81%
Fair (AO03)	19	23.46%
Good (AO04)	8	9.88%
Very Good (AO05)	1	1.23%
No answer	1	1.23%
Not completed or Not displayed	26	32.10%

Summary for G03Q06

How would you rate the availability of open-source IaC deployments that comprehensively manage a complete IT environment?



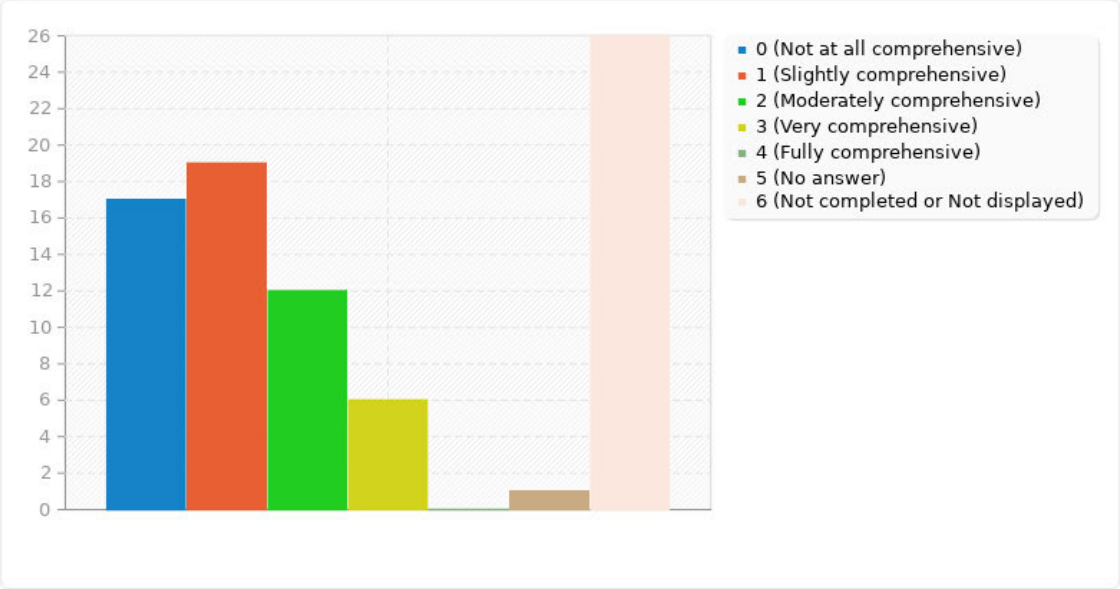
Summary for G03Q07

In your opinion, to what extent do the currently available open-source IaC code examples provide a comprehensive blueprint for managing an entire IT environment?

Answer	Count	Percentage
Not at all comprehensive (AO01)	17	20.99%
Slightly comprehensive (AO02)	19	23.46%
Moderately comprehensive (AO03)	12	14.81%
Very comprehensive (AO04)	6	7.41%
Fully comprehensive (AO05)	0	0.00%
No answer	1	1.23%
Not completed or Not displayed	26	32.10%

Summary for G03Q07

In your opinion, to what extent do the currently available open-source IaC code examples provide a comprehensive blueprint for managing an entire IT environment?



Summary for G03Q08

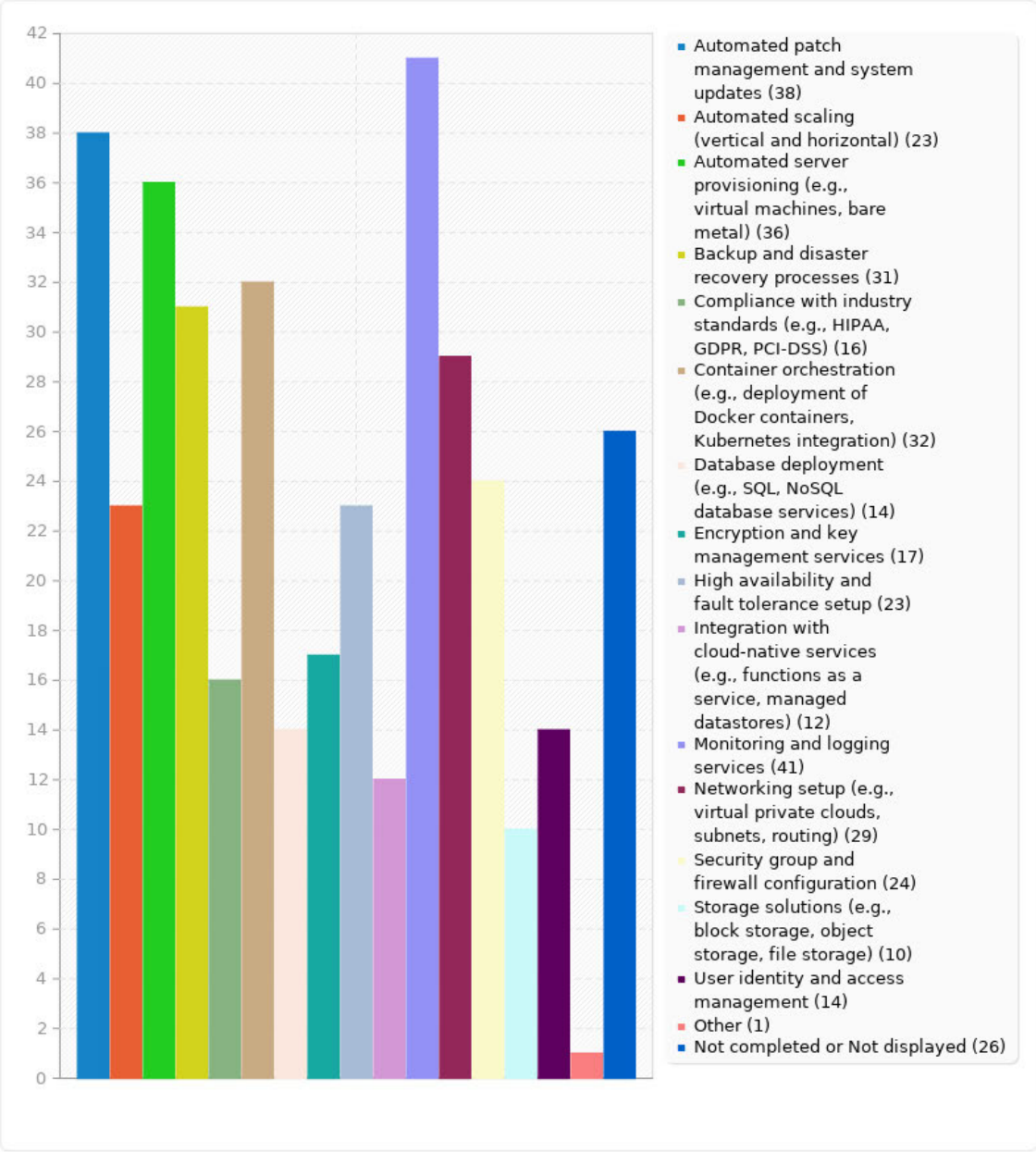
Select the IT environment capabilities you consider most critical to be included in IaC code deployment.

Answer	Count	Percentage
Automated patch management and system updates (SQ001)	38	46.91%
Automated scaling (vertical and horizontal) (SQ002)	23	28.40%
Automated server provisioning (e.g., virtual machines, bare metal) (SQ003)	36	44.44%
Backup and disaster recovery processes (SQ004)	31	38.27%
Compliance with industry standards (e.g., HIPAA, GDPR, PCI-DSS) (SQ005)	16	19.75%
Container orchestration (e.g., deployment of Docker containers, Kubernetes integration) (SQ006)	32	39.51%
Database deployment (e.g., SQL, NoSQL database services) (SQ007)	14	17.28%
Encryption and key management services (SQ008)	17	20.99%
High availability and fault tolerance setup (SQ009)	23	28.40%
Integration with cloud-native services (e.g., functions as a service, managed datastores) (SQ010)	12	14.81%
Monitoring and logging services (SQ011)	41	50.62%
Networking setup (e.g., virtual private clouds, subnets, routing) (SQ012)	29	35.80%
Security group and firewall configuration (SQ013)	24	29.63%
Storage solutions (e.g., block storage, object storage, file storage) (SQ014)	10	12.35%
User identity and access management (SQ015)	14	17.28%
Other	1	1.23%
Not completed or Not displayed	26	32.10%

ID	Response
43	2FA and like

Summary for G03Q08

Select the IT environment capabilities you consider most critical to be included in IaC code deployment.



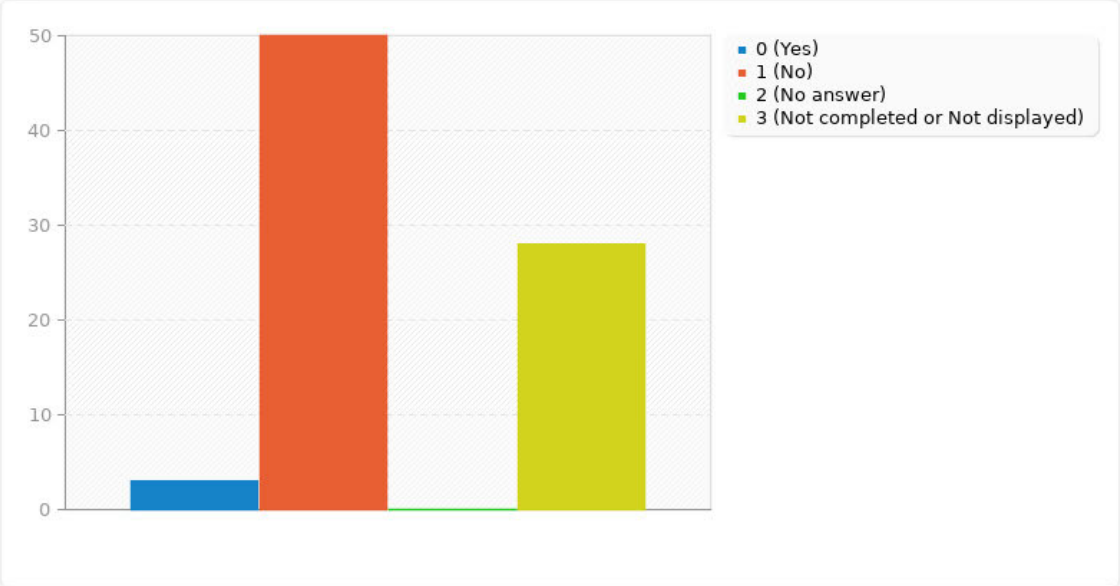
Summary for G04Q08

"Would you like to highlight any important aspects of open-source IaC deployments not covered in this survey?

Answer	Count	Percentage
Yes (Y)	3	3.70%
No (N)	50	61.73%
No answer	0	0.00%
Not completed or Not displayed	28	34.57%

Summary for G04Q08

"Would you like to highlight any important aspects of open-source IaC deployments not covered in this survey?



Summary for G04Q09

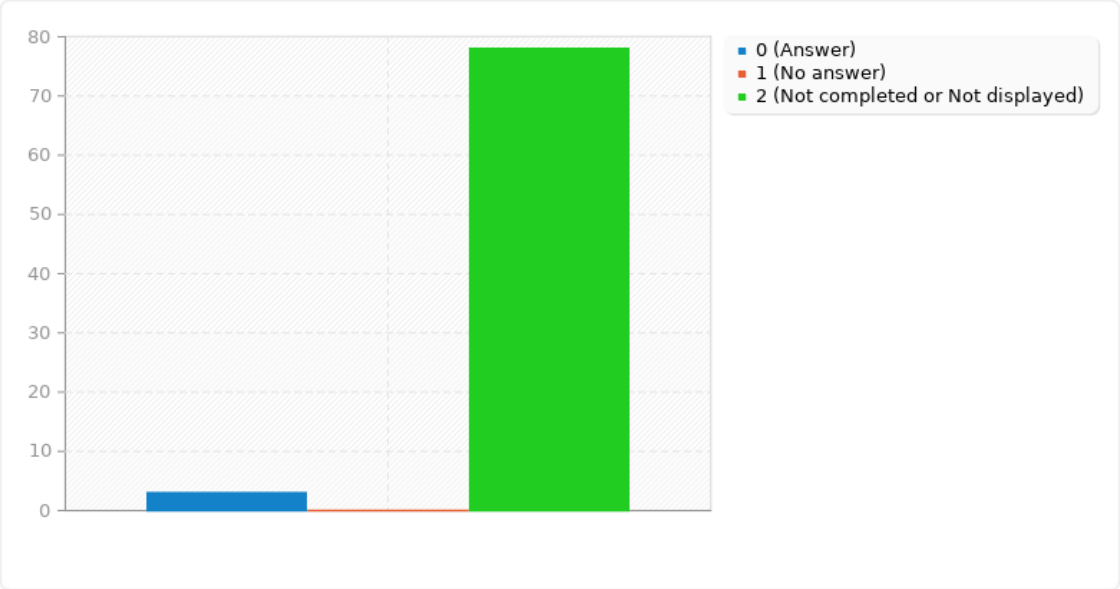
Type your additional thoughts into the textbox below.

Answer	Count	Percentage
Answer	3	3.70%
No answer	0	0.00%
Not completed or Not displayed	78	96.30%

ID	Response
33	IaC, when considered from the beginning or if fully accepted by the higher management is already a complex matter. The most important things to keep in sight is to keep the initial philosophy of all the tools and concepts while make sure everything is well integrated with each other. For this purpose, CMDB is key. To me it is essential that the design is articulated around the CMDB model. Now, when the management is not really aware of the complexity or not fully implicated, the gaol stated above is not easily attainable. I would be delighted to exchange on this domain and your evolution. To give you more context on my experience as it is quite shallow, I have been thinking and proposing such architecture for years and built a wide and comprehensive idea on the subject. However, I have never been able to implement a full automated system.
45	Distinction between open source vs closed source software and free/self-hostable vs paid PaaS.
68	sorry, it's just to comment on question 5: there should be a "not applicable" according to the description but there is no such choice in the form.

Summary for G04Q09

Type your additional thoughts into the textbox below.



Hilfsmittelangabe zum Einsatz von KI-basierten Werkzeugen bei der Anfertigung von wissenschaftlichen Arbeiten

Persönliche Angaben

Sauna, Agostino Fabio	[redacted]
Nachname, Vorname	Matrikelnummer
[redacted]	WIMBIT21B
Anschrift	Kurs
[redacted]	[redacted]
E-Mail	Telefonnummer / Handynummer

Für das Modul Bachelorarbeit
Modulbezeichnung / Semester

muss ich am 08.04.2024
Datum der Frist

folgende Prüfungsleistung erbringen:

☐ Projektarbeit I ☐ Projektarbeit II ☐ Sonstige _____
genaue Bezeichnung

☐ Seminararbeit ☒ Bachelorarbeit

Zur Verwendung KI-gestützter Werkzeuge erkläre ich in Kenntnis des Hinweisblatts "Hinweise zum Einsatz von KI-basierten Werkzeugen bei der Anfertigung wissenschaftlicher Arbeiten und die prüfungsrechtlichen Folgen ihres Einsatzes" Folgendes:

- **Ich habe mich aktiv über die Leistungsfähigkeit und Beschränkungen der in meiner Arbeit eingesetzten KI-Werkzeuge informiert.**
- **Bei der Anfertigung der Prüfungsleistung habe ich durchgehend eigenständig und beim Einsatz KI-gestützter Werkzeuge maßgeblich steuernd gearbeitet.**
- **Insbesondere habe ich alle Inhalte aus wissenschaftlichen oder anderen zugelassenen Quellen entnommen und diese gekennzeichnet sowie alle Inhalte unter Anwendung wissenschaftlicher Methoden selbst entwickelt.**
- **Mir ist bewusst, dass ich als Autor/in der Arbeit die Verantwortung für die in ihr gemachten Angaben und Aussagen trage.**
- **Ich habe keine weiteren als die nachstehend von mir benannten KI-gestützten Werkzeuge zur Erstellung der Arbeit eingesetzt und diese nur in der angegebenen Art und Weise.**

- **Soweit ich KI-gestützte Werkzeuge zur Erstellung der Arbeit eingesetzt habe, gebe ich diese in der nachstehenden "Übersicht verwendeter Hilfsmittel" mit ihrem Produktnamen und ihres genutzten Funktionsumfangs vollständig an; wörtliche oder sinngemäße Übernahmen KI-generierter Inhalte habe ich in ein separates Verzeichnis aufgenommen und im Text belegt (z. B. als Fußnote). Diese Inhalte sind als .pdf-Datei in den elektronischen Beigaben hinterlegt.**

Produktname(n) eingesetzter Hilfsmittel:

LanguageTool (ist eigentliche keine KI wirbt aber damit neuerdings - in Nutzung seit über 3 Jahren)

ChatGPT 3.5 (Gratis Version)

Genutzter Funktionsumfang im Hinblick auf

- **Themenerfassung und Strukturierung** (Aufgabenstellung oder Präzisierung, Forschungsansatz, Gliederung)

Die anfängliche Methodologie war DSRM und die Forschungsfrage standen sehr früh fest. ChatGPT wurde gebeten basierend auf die Forschungsfrage und der Methodologie mein Vorgehen strengstens zu kritisieren. Daraufhin kam das Feedback, dass die Case-Study Umgebung evtl nicht Praxisrelevant sei. ChatGPT hat keine guten Lösungsvorschläge gemacht, aber die Kritik war der Impuls für mich die Tabelle mit Funktionen von IT Umgebungen zu erstellen und das über Interviews absegnen zu lassen.

- **Themenbearbeitung** (Darstellung/Beschreibung des Problems, Darlegung von wissenschaftlichen Grundlagen, Schlussfolgerungen allgemein und für das konkrete Thema / Erkenntnisgewinn, Evaluierung des Textes / Feedback der KI)

Zu Beginn der Recherche wurde ChatGPT genutzt um schwerverstehende Passagen von Papern erklären zu lassen. Die Erklärungen dienten lediglich zum Verständnis und wurden daraufhin mit präziseren Google Suchen auf Echtheit verifiziert. Thematisch ging es um die Abgrenzung von DevOps, CI/CD und IaC

- **Quellenrecherche, -auswahl und -auswertung** (welche Quellen wurden durch das KI-Werkzeug gefunden, wie erfolgte die weitere Recherche)

- **Formale Gestaltung, insbesondere Sprache** (Welche Eingabe wurde an die KI getätigt? Textgenerierung, Textkorrektur, Paraphrasieren und Umschreiben, Übersetzen, kreatives Schreiben)

LanguageTool um Grammatik-, Rechtschreib-, Komma- und Stilfehler (wiederholende Begriffe) zu finden. Von dem Funktionsumfang handelt es sich wie die Rechtschreibprüfung von Word, aber in besser.

Hinweis:

Geben Sie in der jeweiligen Rubrik die Seitenzahl in ihrer wissenschaftlichen Arbeit an und erläutern Sie die Art und Weise sowie den Umfang der von Ihnen genutzten KI-basierten Werkzeuge.

Mannheim, 06.04.2024 [redacted]

Ort, Datum **dierenden**